HARK Cookbook

HARK Documentation Team

March 16, 2011

Contents

1 Introduction **4**
1.1 First sound recording . . . 4
1.2 First sound localization . . . 7
1.3 First sound separation . . . 8
1.4 First sound recognition . . . 9

2 Something is wrong **10**
2.1 The installation cannot not be performed well, what should I do? . . . 10
2.2 The sound recording cannot be performed well. What should I do? . . . 12
2.3 The localization cannot be performed well. What should I do? . . . 13
2.4 The separation cannot be performed well. What should I do? . . . 14
2.5 The recognition cannot be performed well. What should I do? . . . 15

3 Microphone array **16**
3.1 How many microphones should be used? . . . 16
3.2 How should I arrange microphones? . . . 17
3.3 What types of microphones used should I use? . . . 18
3.4 What should I have to be careful with when I set microphones on the robot? . . . 19
3.5 What sampling rate should I use? . . . 20
3.6 How should I use other types of A/D converters? . . . 21

4 Input data generation **26**
4.1 Multi-channel recording . . . 26
4.2 Microphone connection check . . . 27
4.3 Sound recording with TED 16ch device . . . 28
4.4 Impulse response recording . . . 30
4.5 Synthesis of 8ch data . . . 31
4.6 Addition of noise data . . . 32

5 Acoustic model and language model **33**
5.1 How should I create acoustic model? . . . 33
5.2 How should I create language model? . . . 41

6 FlowDesigner **43**
6.1 Startup with batch . . . 43

7 Sound source localization **45**
7.1 How should I perform sound source localization? . . . 45
7.2 How should I create A matrix from the transfer function measured? . . . 47
7.3 How should I count the number of microphones? . . . 48
7.4 How many sound sources should I use for LocalizeMUSIC? . . . 49
7.5 Check if the sound source is localized successfully . . . 50
7.6 How should I determine the threshold values for SourceTracker? . . . 51

7.7 How should I determine PAUSE_LENGTH for SourceTracker? . . . 52
7.8 How should I use SourceIntervalExtender? . . . 53
7.9 How should I perform two- or three-dimensional localization? . . . 54
7.10 How should I save localization results in files? . . . 55

8 Separation **56**
8.1 How should I separate sounds? . . . 56
8.2 How should I save separated sounds in files? . . . 57
8.3 GHDSS: How should I create transfer functions? . . . 58
8.4 GHDSS: How should I designate microphone position? . . . 59
8.5 GHDSS: How should I designate file of robot noise? . . . 60
8.6 GHDSS: How should I set step size? . . . 61
8.7 GHDSS: How should I set UPDATE_METHOD_TF_CONJ, UPDATE_METHOD_W? . . . 62
8.8 GHDSS: How should I tune? . . . 63
8.9 PostFilter: How should I tune? . . . 64

9 Feature extraction **65**
9.1 What types of features should I use? . . . 65
9.2 Type of feature . . . 66
9.3 How should I save features in files ? . . . 67
9.4 How should I set threshold value for MFT? . . . 68

10 Recognition **69**
10.1 .How should I set jconf file? . . . 69

11 Advanced usage **71**
11.1 How should I create nodes? . . . 71
11.2 How should I improve the system performance?: Processing speed . . . 92
11.3 How should I improve the system performance?: Sound source localization . . . 93
11.4 How should I improve the system performance?: Sound source separation . . . 95
11.5 How should I create debug module? . . . 97
11.6 How should I use debug tool? . . . 98
11.7 How should I connect with other systems by TCP/IP? . . . 99
11.8 How should I connect with ROS? . . . 107
11.9 How should I control motor? . . . 108
11.10 How should I copy nodes from other network files? . . . 109

12 Others **110**
12.1 What window length and shift length should I use? . . . 110
12.2 Types of windows used for MultiFFT . . . 111
12.3 What is MAP? . . . 112
12.4 How should I use PreEmphasis? . . . 113

13 Recipes **114**
13.1 Category of sample network . . . 114
13.2 Outline of sample network category . . . 114
13.3 Notation of document and method for execution of sample network . . . 115
13.4 Sound recording network sample . . . 115
13.4.1 Monaural sound recording . . . 115
13.4.2 Stereo recording . . . 117
13.4.3 8ch sound recording with radio RASP . . . 118
13.5 Network sample of sound source localization . . . 121
13.5.1 Off line sound source localization . . . 121
13.5.2 Online sound source localization . . . 122
13.6 Network sample of sound source separation . . . 125

13.6.1 Off line sound source separation . . . 125
13.6.2 Online sound source separation . . . 127
13.6.3 Off-line sound source separation (with postprocessing by HRLE) . . . 128
13.7 Acoustic feature extraction network sample . . . 129
13.7.1 MSLS feature extraction (without delta term without power term) . . . 132
13.7.2 MSLS feature extraction (with delta term, without power term) . . . 133
13.7.3 MSLS feature extraction (without delta term, with power term) . . . 134
13.7.4 MSLS feature extraction (with delta term, with power term, with delta power term) . . . 135
13.7.5 MSLS feature extraction (with delta term, without power term, with delta power term) . . . 138
13.7.6 MSLS feature extraction (with pre-emphasis, with mean subtration, with delta term, without power term, with delta power term) . . . 138
13.8 Speech recognition network sample . . . 141
13.8.1 Separated sound recognition processing based on MFT with MSLS . . . 142

Chapter 1

Introduction

We describe FAQ for using HARK.

1.1 First sound recording

Start with setting 1ch recording

The first sound recording with HARK. Start with sound recording of 1ch.

Solution

Deepen understanding while executing a demo. Execute demoALSA1ch.sh in the Record directory of Sample. After the execution, the sound is recorded for about 1 second, and a file named rec0.sw.wav is generated. Replay the file and confirm the sound recorded. When recording cannot be performed well, execute demoALSA1ch.sh again after confirming each of the following items.

(1) Confirm if the microphone is connected properly.
Confirm if the plug is plugged or not and connect it properly.

(2) Confirm if the PC accepts plug-in power.
When plug-in power is supported, the power is supplied to the microphone from the PC side. However, when plug-in power is unsupported, it is necessary to supply power to the microphone. Use battery box. Connect the battery box and turn it on.

(3) Confirm if sound recording software except HARK can be used to replay or record sounds.
When such software cannot replay or record sounds, setting of the OS and driver may be wrong, or the audio interface is not set or connected properly.

(4) When connecting more than two audio interfaces, remove the ones from the second, and confirm if it can record sounds.
When a sound cannot be recorded with one connection audio interface, change properties of demo.n. Set plughw:0,0, plughw:0,1 or plughw:0,2 for DEVICE property of the AudioStreamFromMicmodule and try sound recording.

(5) In the case that rec0.sw.wav is not generated though it seems to work normally, the cause is that SoX (http://sox.sourceforge.net/) has not installed yet. Therefore, install SoX.Move to http://sox.sourceforge.net/ Sound eXchange, which is the link to associated network files.

Discussion

This sample consists of five modules. There is one module in MAINLOOP (iterator) and four modules in MAIN (subnet). MAIN (subnet) and MAINLOOP (iterator) are shown in Figures 1.1 and 1.2. Opening the property

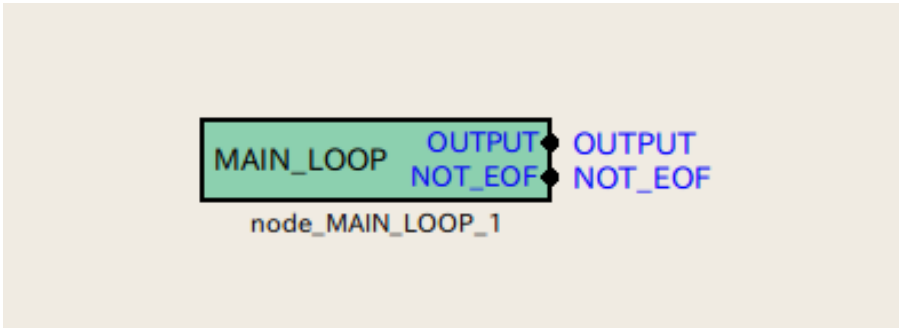


Figure 1.1: MAIN (subnet)

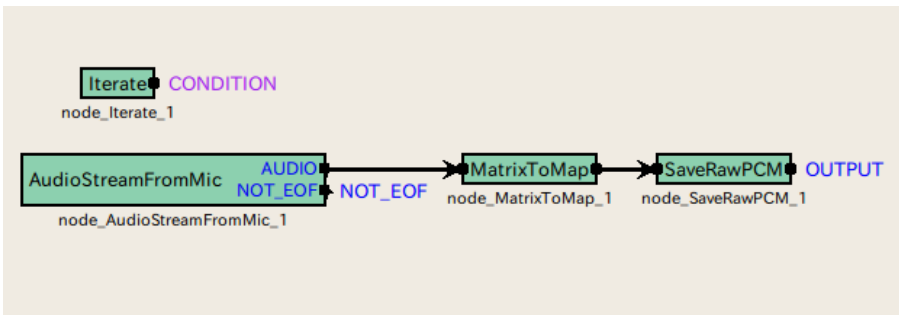


Figure 1.2: MAINLOOP (iterator)

of the MAINLOOP module in MAIN (subnet), the user can confirm that this module has five parameters. Table 1.1 shows the parameters. The parameters important for this demo are SAMPLING_RATE and GETFRAMES. These parameters are the sound sampling frequency for recording and the number of frames to be acquired. Sound recording time length is designated by the number of frames to be acquired. Actual sound recording time length by sec is expressed as follows.

$$(LENGTH + (GETFRAMES - 1) * ADVANCE)/SAMPLINGRATE \tag{1.1}$$

Although sound recording time length depends on LENGTH and ADVANCE, it is not necessary to change these parameters for this demo. Change sampling frequency and the acquired number of frames and try to record sounds in a desired sound recording format. However, when changing the sampling frequency, change 16000 of fs in demo.sh to the desired sampling frequency. Moreover, when changing the acquired number of frames, change 100 of nframe to the desired number of frames. Among the four modules in MAINLOOP (iterator), the important

Table 1.1: Parameter list of MAINLOOP

Parameter name	Type	Default value	Unit	Description
ADVANCE	`int`	160	[pt]	Shift length
LENGTH	`int`	512	[pt]	FFT length
SAMPLING_RATE	`int`	16000	[Hz]	Sampling frequency
GETFRAMES	`int`	`int`:ARG1		The number of frame for sound recording
DOWHILE	`bool`			Leave blank

module is AudioStreamFromMic. Designate a sound recording device in this module. For this demo, sound recording channel is only 1ch so that more operations can be checked and plughw:0,0 is set for DEVICETYPE ALSA and DEVICE.

Table 1.2: Parameter list of AudioStreamFromMic

Parameter name	Type	Default value	Unit	Description
LENGTH	subnetparam	LENGTH	[pt]	FFT length
ADVANCE	subnetparam	ADVANCE	[pt]	Shift length
CHANNEL_COUNT	`int`	1	[ch]	Number of sound recording channels
SAMPLING_RATE	subnetparam	16000	[Hz]	Sampling frequency
DEVICETYPE	`string`	ALSA		Device type
DEVICE	`string`	plughw:0,0		Device name

See Also

None.

1.2 First sound localization

Problem

I want to perform source localization in HARK but don't know what to start with.

Solution

(1) Source localization of audio file

First, create a system that displays localization result with sounds already recorded. Sample files of source localization are provided. So deepen understanding executing them. In order to simply operate, execute demo.sh in the Localize directory of the sample files. A localization result of the zero-degree direction is indicated first and then localization results are shown for 90, 180 and -90 degrees.

(2) Real time source localization from microphone

Next, record sounds from microphones, and create a real-time source localization system. The creation method of a network for source localization is described in detail in Section 2.3 of the HARK document. Please read them. When it does not work, read "The sound cannot be recorded well. What should I do?" or "The localization cannot be performed well. What should I do?".

Discussion

The information of acoustic directions is easy-to-use and the source localization network is also simple so it is an optimal robot audition technique to be tested first. Source localization can be performed almost certainly in an ordinary room with no noise where an utterer speaks loudly but not with cracked sounds, close to the microphone array. The user can confirm that if the utterer walks slowly while speaking, localization results follow it. To raise the accuracy more, read the recipe in Chapter 7 or descriptions of the modules LocalizeMUSICand SourceTrackerin the HARK document and tune it.

See Also

"The sound cannot be recorded well. What should I do?"
"The localization cannot be performed well. What should I do?"
The sample files are available at the following URL.

HARK Main page (http://winnie.kuis.kyoto-u.ac.jp/HARK/)

1.3 First sound separation

Problem

First sound source separation with HARK. First, separate only the acoustic signals from a specific direction from the sound recorded.

Solution

demoOfflineHolophone.sh in the Separation directory of SampleKit performs sound source separation. This section describes the method to separate sound sources in an example with this sample. The following files are required to perform demoOfflineHolophone.sh.

1. demoOfflineHolophone.n
2. 10wordsHolophone.wav
3. GHDSSHolophone.dat

When executing this sample, only the utterance of a speaker in the front among three speakers (front, 90 degrees in left, 90 degrees in right) is separated and recorded as sep0.wav (When there is already sep0.wav, the index increases like sep1.wav).

Discussion

In the sample of sound source separation, only the speech from the front of the 7ch speech data is separated and the separated sound is saved. Each parameter is tuned beforehand for the sample so the sound source separation power may deteriorate in different environments. For the tuning method, see "Separation".

See Also

When separation cannot be performed well, see "The separation cannot not be performed well. What should I do?. Since sound source separation is performed after source localization, it is important to confirm if the stages before sound recording or source localization are performed properly. For sound recording and source localization, "First sound recording", "First source localization", "The sound recording cannot be performed well. What should I do?" and "The localization cannot not be performed well. What should I do? will be helpful.

1.4 First sound recognition

Problem

First speech recognition with HARK. First, read data from an audio file and perform speech recognition.

Solution

Execute a demo to deepen understanding. Execute demorecognition.sh in the Recognition directory of SampleKit. This sample starts the network and the speech recognition engine Julius of HARK. HARK reads data of audio files, extracts MSLS features and sends the features to the speech recognition engine Julius. Julius receives and recognizes features, and displays results in standard outputs. When recognition is not performed well, read "The recognition cannot be performed well. What should I do?".

Discussion

None.

See Also

None.

Chapter 2

Something is wrong

2.1 The installation cannot not be performed well, what should I do?

Problem

Read this recipe when installation cannot be not performed by the following two standard methods.

1. Method with the simplest debian package
   Download the .deb file from the web site of HARK and execute the following command at the directory where there is the file.

   ```
   dpkg -i ‘‘a hark packakge’’
   ```

2. Compile from the source
   Compilation and installation are performed with the familiar command.

   ```
   ./configure
   make
   make install
   ```

Solution

It is often the case that an old version HARK or the wreck generated at the failure of HARK installation in the past are the cause. Completely remove the HARK from your system and install again.

Uninstall all the HARK-related software. If installing by dpkg, remove it by the following command.

```
dpkg -P packagename.deb
```

If compiling from the source, execute the following command to delete it.

```
make uninstall # remove copied files
make clean
# remove compiled files
make distclean # remove Makefile
```

1. Check if the old HARK is deleted properly.
   Execute `flowdesigner` in a command line. It is successfully deleted if the error ”The command is not found” is displayed.
   Execute `ls /usr/local/lib/flowdesigner` and confirm if the directory is deleted. If there is, delete it. If another directory is designated for the past compilation, confirm it and delete it if there is.

2. Install again. Perform installation in the following order.
   (1) flowdesigner, (2) libharkio.deb (3) hark-fd, (4) hark-tool.

Discussion

It is often the case that severe problems in installation are caused by the following two:

1. Dependencies between software has not been solved.
2. The path for installation is different.

This recipe introduces a method to confirm from the beginning. However, it is important to read the error message well before everything else. It's no exaggeration to say that a half the problem is solved if the use understands the error messages.

See Also

When the installation is completed successfully, read from "First HARK".

2.2 The sound recording cannot be performed well. What should I do?

Problem

It is often the case that the reason why a system does not work well is because recording itself is not performed properly. This section describes

1. a method to confirm if the recording is performed properly and

2. the typical reason why sound recording is not performed and measures for it.

Solution

Try to record while reading Recipe 2.1 first. It will be easier to confirm later if the user speaks something for sound recording. Next, confirm the file recorded using the software that can display waveforms. For example, Audacity and wavsurfer can display waveforms. When recording with more than three microphones, it is convenient to use software that accepts those plural channels. As a result of displaying, if the own waveforms are recorded for all channels to which microphones are connected, the recording has been successfully completed. Connection is right at least. In the case that there are almost no amplitudes or clipping occurs, confirm the following.

1. Isn't each microphone out of order?
   Connect them to a familiar PC and confirm if each of them can record sounds one by one.

2. Are the connectors of the microphones connected properly?
   Confirm if the connectors are connected.

3. Is plug in power used?
   Some types of microphones cannot secure sufficient sound volume without external power sources (plug in power). Although some sound recording devices supply power, in the case that the microphone needs plug in power and when there is not a device to supply power, it is required to supply power. Confirm manuals of both the microphone and device.

Discussion

It is often the case that the speech recognition system does not work because the sound recording itself was not performed properly. In particular, for a system with a large number of microphones, it is important to confirm that each microphone can record sounds steadily before operating software.

See Also

The detailed method of sound recording is described in "First sound recording" in Recipe 2.2 and sound recording devices recommended for HARK are described in Chapter 7 in hark-document.

2.3 The localization cannot be performed well. What should I do?

Problem

Read this section when although a source location system is created but localization is not performed well. This section describes the causes and measures for inoperative of typical localization systems.

Solution

First, confirm if the sound recording is completed successfully. Read Recipe 3.2 and confirm if sound recording can be performed. If sound recording was successfully completed, test if localization can be performed with a waveform file, not microphones. A waveform file is obtained by recording with the microphone array created. For example, record sounds for 10 seconds while walking slowly around the microphone array speaking alone. Confirm if localization is performed with such a waveform file. Localization results can be confirmed visually by connecting DisplayLocalizationto the output of LocalizeMUSICin the network. If localization is not performed well at this stage, adjust parameters of LocalizeMUSIC.

1. Set NUM_SOURCE for the condition that localization system is intended for.
   For example, if presuming that there are up to two speakers, set NUM_SOURCE to 2.

2. Set MIN_DEG and MAX_DEG.
   In the condition that localization results are obtained though there is not a sound source, there may be reflection from the wall and a noise source (e.g. fans of a PC). In such a case, if it can be presumed that there is not the target sound source in the direction, the user can set so that localization results are not output from the direction with these two parameters. For example, supposing that there will not be the case that somebody talk to the from behind the robot, the localization target will be only the front side by setting MIN_DEG to -90 and MAX_DEG to 90.

Next, adjust the parameter THRESH of SourceTracker. Sound sources with the power smaller than THRESH are ignored and therefore a good performance is obtained by adjusting it for the environment well. In order to achieve it, set the DEBUG property of LocalizeMUSICto `true`first, and observe the value that enables to output MUSIC spectrums (power of each direction) during the execution. Set a value that is smaller by around 0.1 - 0.2 than the comparatively great value seen in the sound source direction (around 27-30) to THRSH. If the operation is successfully performed with the waveform file, operate the system with actual microphones. Since the appropriate value of THRESH changes depending on environments and the number of speakers, if there are too many error localization results, increase THRESH a little and if localization results are not obtained with speech, decrease THRESH a little as tuning.

Discussion

The appropriate parameter changes depending on reverbation in the room and the voice volume of a speaker. In order to raise localization performance in the current system, it is better to tune up in the actual scenes. Since THRESH is the most important parameter here, adjust only it basically.

See Also

When creating a localization system for the first time, see Recipe 2.2. Moreover, for the details of the modules important for source localization, see LocalizeMUSICand SourceTrackerof hark-document.

2.4 The separation cannot be performed well. What should I do?

Problem

Read this section when a sound that cannot be considered speech is output or heavy distortion occurs, in intension to separate sound mixture from a microphone input.

Solution

When separating with the GHDSSmodule, perform the separation in accordance with the following conditions.

1. Confirm if the source localization is successfully completed Since the GHDSSmodule uses source localization results (number of sound sources) for inputs, its original separation power is not obtained if the localization ended in fail. Set so that network files only for source localization or localization results are displayed and confirm if the localization finished in success. In order to deal with this, see The localization cannot be performed well. What should I do?

2. Confirm if the output from GHDSSis separated In order to deal with the case that the processing of GHDSSitself is not performed properly, output a result after GHDSSfirst and confirm if the separation is finished in success. When the separation ended in fail in GHDSS, see "GHDSS: How should I tune?".

3. Confirm if Postfilter is performed properly When Postfilter is inserted just after separation processing, separation may fail due to improper parameter values. In such a case, see "Postfilter: How should I tune?".

Discussion

None.

See Also

"The localization cannot be performed well. What should I do?",
"GHDSS: How should I tune?",
"PostFilter: How should I tune?".

2.5 The recognition cannot be performed well. What should I do?

Problem

I am making a speech recognition system with HARK but it does not recognize any speech.

Solution

If you haven't tried the methods in Section (1) in the tutorial, starting with this section will be an easier way for you. In this tutorial, the user learns about a speech recognition system with HARK in order of sound recording, sound source localization, sound source separation and speech recognition. Next, an inspection method for the case that an original system is developed by the user and it does not work is described. Since a large number of elements are included in a speech recognition system, it is important to verify possible causes one by one. First, confirm if HARK and FlowDesigner are installed properly (Recipe 2.1), if sound recording is performed properly (Recipe 2.2), if sound source localization is performed properly (Recipe 2.3) and if separation is performed properly (Recipe 2.4) in each recipe. Verifying the system till this stage, if you find that the system works properly, verify speech recognition. Here, it is premised that Julius (http://julius.sourceforge.jp/), the large vocabulary continuous speech recognition engine, which has been modified for HARK, is used for the system.

The following three files are important for using Julius.

1. Acoustic model: A model to indicate relations between features and phonemes of acoustic signals
2. Language model: A model of the language that the system user speaks
3. Configuration file: File names of the above two files

For the case that the system does not work at all or Julius does not start, it is often the case that a wrong path is designated for the file so it is easier for you to check the configuration file. For details, see Recipe 10.1. Or, when you verify the model itself, see 5.1 for the acoustic model or see 5.2 for the language model.

Discussion

Above are measures for the case that the system does not recognize sounds at all. For the case that "The system works properly but success rate (= recognition rate) is bad", you need to tune up. Speech recognition system tuning involves complicated problems of localization / separation / recognition and so on. For example, see parameter tuning of GHDSS(Recipe 8.8) for separation and tuning of PostFilter(Recipe 8.9). The chapter that discusses features used for recognition 9 and recipes for improving performance (11.2, 11.3, 11.4) are also helpful.

See Also

Recipes only for the case that the system does not recognize in the first place are listed here.

1. Each recipe in Chapter 2 "Something is wrong"
2. Acoustic model (Recipe 5.1) and language model (Recipe 5.2)
3. Configuration file of Julius (Recipe 10.1)
4. Document of Julius

Chapter 3

Microphone array

3.1 How many microphones should be used?

Problem

Read this section when mounting microphones on your robot.

Solution

Theoretically, N sound sources can be separated with "the number of sound sources + 1" pieces of microphones. In fact, its performance depends on the layout / operating environment of individual microphones. It is rare that the number of sound sources is already known so the best number of microphones is determined by trial and error. Select the minimum number with which the performance of sound source localization / sound source separation / separated sound recognition reaches the required level. The theoretically best layout for treating sound sources of all directions is an equally-spaced layout on the arc of a concentric circle. Here, it is premised that the head of the robot is a complete sphere. If it were not a complete sphere, it might cause performance degradation of sound source localization and sound source separation in a direction with the head shape that occurs discontinuous reflection. It is better to set a concentric circle in the head where continuous reflection occurs.

Discussion

Presently, in our three-speaker simultaneous utterance recognition demonstration, three sound sources are separated with eight microphones. The number of separatable sound sources is below a half the theoretical value 7. It is possible to increase the number of microphones, for example, to sixteen so as to improve performance and separate a large number of sound sources. However, when the microphone interval becomes too close, it makes it difficult to improve performance and calculation costs increase.

See Also

None.

3.2 How should I arrange microphones?

Problem

Read this section when mounting microphones on your robot.

Solution

The precondition for layout is that relative positions between individual microphones do not vary. The layout for which localization / separation accuracies are considered is required and appropriate layout depends on directions of sound sources as a processing object. When a sound source from a specific direction can be assumed, position the microphones closely in such a direction. It is better to disperse microphones widely perpendicular to the normal vector of the specific direction. When treating sound sources of all directions, position them in a circular pattern. A wider microphone interval is more profitable for sound source separation. Position microphones at an as wide interval as possible. At least, in a layout that sound source localization cannot be performed, it is difficult to perform sound source separation. Therefore, check if sound source localization can be performed to determine a layout. When localization accuracy is not good, check if the layout is not in the shape by which acoustic reflection becomes discontinuous around microphone positions, and avoid such a location for positioning.

Discussion

When a relative position between microphones varies, an impulse response of a microphone array varies. It is premised that impulse responses of a microphone array are fixed in HARK.

See Also

None.

3.3 What types of microphones used should I use?

Problem

Read this section when mounting microphones on your robot.

Solution

We use non-directional electret condenser microphones that cost only several thousand yen. It is not always necessary to use expensive microphones. Microphones with a great signal-to-noise ratio are preferable. It is better to choose wires with proper covering.

Discussion

In order to record sounds with a sufficiently great signal-to-noise ratio, choose wires with proper covering and with plugs. Although such a microphone usually is expensive, we have already confirmed that the system works with microphones of around several thousand yen. When wiring inside the robot, pay attention to covering of wires and suppression of interference by other signal wires, rather than using expensive microphones.

See Also

None.

3.4 What should I have to be careful with when I set microphones on the robot?

Problem

Read this section when mounting microphones on your robot.

Solution

For wiring of microphones, shorten wire lengths of them and for wiring inside the robot, do not wire in parallel with other wiring such as power wires and signal wires for servos. For signal transmission, it is better to use differential and digital types. Set microphones so that they contact the surface of the robot's housing. In other words, set microphones so that they are embedded in the housing and only the microphone tips are opened outside.

Discussion

For wiring of microphones, shorten wire lengths of them and for wiring inside the robot, do not wire in parallel with other wiring such as power wires and signal wires for servos, not to pick up noise from other signal wires. Such consideration is important for single end in particular. Set microphones so that they contact the surface of the robot's housing. In other words, set microphones so that they are embedded in the housing and only the microphone tips are opened outside. When microphones are isolated from the surface, they are affected by reflection from the housing, which leads to degradation of sound source localization and sound source separation performances. When the housing vibrates in robot operation, consider so that microphones do not pick up such vibration. Use materials that suppress vibration as a mount, such as a bush.

See Also

None.

3.5 What sampling rate should I use?

Problem

Read this section when the user does not know how to determine a sampling rate in entering acoustic signals from a device.

Solution

When there are no particular reasons, 16kHz is good enough for a sampling rate. When there are, use the lowest sampling rate within the range that alias does not occur. When setting a sampling rate, it is necessary to consider so that alias does not occur. The frequency that is a half of a sampling rate is called Nyquist frequency. Alias occurs in a signal with a frequency component over this Nyquist frequency. Therefore, in order to prevent occurrence of alias in input signals, it is preferable to use an as high sampling rate as possible. When the highest sampling rate is selected, alias can most hardly occur. On the other hand, when raising a sampling rate, amounts of data to be processed increase, which leads to an increase of calculation costs. Therefore, when setting a sampling rate, it is better not to raise a sampling rate by more than necessary.

Discussion

Speech energy reaches over 10kHz in bandwidth and the energy is present a lot in low frequency components. Therefore, it is often the case that consideration of low frequency bands is sufficient for most purposes. For example, in the case of telephone, it is from around 500Hz to 3,500Hz and therefore it is possible to transmit interpretable audio signals with a bandwidth up to around 5kHz [1]. In the case of the 16kHz sampling, frequency components equal to or less than 8kHz can be sampled without alias and therefore it is used often for speech recognition. [1] Acoustic analysis of speech, by Ray D Kent and Charles Read, translation supervised by Takayuki Arai and Tsutomu Sugawara, Kaibundo, 2004.

See Also

None.

3.6 How should I use other types of A/D converters?

Problem

Read this section when wishing to capture acoustic signals using devices other than the sound cards supported by ALSA(Advanced Linux Sound Architecture), the RASP series made by System In Fronteir, Inc. and TD-BD-16ADUSB made by Tokyo Electron Device, which are supported by HARK as a standard

Solution

There are two methods to capture acoustic signals with devices other than the three mentioned above. The first one is a method with the HARK 3rd party version and the second one is that the user creates a module corresponding to the device by themselves.

1. Three types of devices are supported for HARK as a standard. An A/D converter can be used by a corresponding node created by the user themselves. Here, a procedure is described with creation of the AudioStreamFromMicnode that supports the new device NewDevice as an example. A general creation procedure is as follows.

   (a) Create the class NewDeviceRecorder corresponding to the device and put its source file and header file in the librecorder directory in the HARK directory.

   (b) Rewrite AudioStreamFromMicso that the created class can be used.

   (c) Rewrite Makefile.am, configure.in to compile.

   In (a), create a class that takes data from the device and send them to a buffer. This class is to be implemented in succession to the Recorder class. Initialize, which performs preparations for using the device and the operator() method that takes out data from the device, is implemented. In (b), processing to be performed when "NewDevice" is designated in the option is described. Concretely, declare and initialize NewDevice in the constructer and set signals. In (c), change Makefile.am, configure.in. in correspondence with the file newly added.

Described below is a sample of AudioStreamFromMic2, which supports a new device (NewDevice)

```
#include "BufferedNode.h"
#include "Buffer.h"
#include "Vector.h"
#include <climits>
#include <csignal>
#
include <NewDeviceRecorder.hpp> // Point1:
Read a required header file
using namespace std;
using namespace FD;
class AudioStreamFromMic2;
DECLARE_NODE( AudioStreamFromMic2);
/*Node
*
* @name AudioStreamFromMic2
* @category MyHARK
* @description This node captures an audio stream using microphones and outputs frames.
*
* @output_name AUDIO
* @output_type Matrix<float>
* @output_description Windowed wave form.
A row index is a channel, and a column index is time.
*
* @output_name NOT_EOF
* @output_type bool
```

```
* @output_description True if we haven't reach the end of file yet.
*
* @parameter_name LENGTH
* @parameter_type int
* @parameter_value 512
* @parameter_description The length of a frame in one channel (in samples).
*
* @parameter_name ADVANCE
* @parameter_type int
* @parameter_value 160
* @parameter_description The shift length between adjacent frames (in samples).
*
* @parameter_name CHANNEL_COUNT
* @parameter_type int
* @parameter_value 16
* @parameter_description The number of channels.
*
* @parameter_name SAMPLING_RATE
* @parameter_type int
* @parameter_value 16000
* @parameter_description Sampling rate (Hz).
*
*
@parameter_name DEVICETYPE // Point2-1:
Add the type of a device to be used
* @parameter_type string
* @parameter_value NewDevice
* @parameter_description Device type.
*
*
@parameter_name DEVICE // Point2-2:
Add the name of a device to be used
* @parameter_type string
* @parameter_value /dev/newdevice
* @parameter_description The name of device.
END*/
// Point3:
Describe processing to stop sound recording in the middle
void sigint_handler_newdevice(int s)
{Recorder* recorder = NewDeviceRecorder::
GetInstance();
recorder->Stop();
exit(0);
}
class AudioStreamFromMic2:
public BufferedNode {
int audioID;
int eofID;
int length;
int advance;
int channel_count;
int sampling_rate;
string device_type;
string device;
Recorder* recorder;
vector<short> buffer;
public:
AudioStreamFromMic2(string nodeName, ParameterSet params)
:
BufferedNode(nodeName, params), recorder(0)
{
audioID = addOutput("AUDIO");
eofID = addOutput("NOT_EOF");
length = dereference_cast<int> (parameters.get("LENGTH"));
advance = dereference_cast<int> (parameters.get("ADVANCE"));
channel_count = dereference_cast<int> (parameters.get("CHANNEL_COUNT"));
sampling_rate = dereference_cast<int> (parameters.get("SAMPLING_RATE"));
device_type = object_cast<String> (parameters.get("DEVICETYPE"));
```

```
device = object_cast<String> (parameters.get("DEVICE"));
// Point4:
Create a recorder class corresponding to the device type
if (device_type == "NewDevice")
{recorder = NewDeviceRecorder::
GetInstance();
recorder->Initialize(device.c_str(), channel_count, sampling_rate, length * 1024);} else {
throw new NodeException(NULL, string("Device type " + device_type + " is not supported."), __FILE__, __LINE__);}
inOrder = true;}
virtual void initialize()
{outputs[audioID].
lookAhead = outputs[eofID].
lookAhead = 1 + max(outputs[audioID].
lookAhead, outputs[eofID].
lookAhead);
this->BufferedNode::
initialize();}
virtual void stop()
{recorder->Stop();}
void calculate(int output_id, int count, Buffer &out)
{Buffer &audioBuffer = *(outputs[audioID].
buffer);
Buffer &eofBuffer = *(outputs[eofID].
buffer);
eofBuffer[count]
= TrueObject;
RCPtr<Matrix<float> > outputp(new Matrix<float> (channel_count, length));
audioBuffer[count]
= outputp;
Matrix<float>& output = *outputp;
if (count == 0)
{ //Done only the first time
recorder->Start();
buffer.resize(length * channel_count);
Recorder::
BUFFER_STATE state;
do {usleep(5000);
state = recorder->ReadBuffer(0, length, buffer.begin());} while (state != Recorder::
OK);
// original
convertVectorToMatrix(buffer, output, 0);} else { // Normal case (not at start of file)
if (advance < length)
{Matrix<float>& previous = object_cast<Matrix<float> > (
for (int c = 0;
c < length - advance;
c++)
{for (int r = 0;r < output.nrows();r++)
{output(r, c)= previous(r, c + advance);}}} else {for (int c = 0;c < length - advance;c++)
{for (int r = 0;r < output.nrows();
r++)
{output(r, c)= 0;}}}
buffer.resize(advance * channel_count);
Recorder::
BUFFER_STATE state;
for (;;)
{
state = recorder->ReadBuffer((count - 1)
* advance + length,
advance, buffer.begin());
if (state == Recorder::
OK)
{break;} else {usleep(5000);}
}
int first_output = length - advance;
convertVectorToMatrix(buffer, output, first_output);
}
bool is_clipping = false;
for (int i = 0;
```

```
i < buffer.size();
i++)
{
if (!is_clipping && checkClipping(buffer[i]))
{
is_clipping = true;
}
}
if (is_clipping)
{
cerr << "[" << count << "][" << getName()
<< "] clipping" << endl;
}
}
protected:
void convertVectorToMatrix(const vector<short>& in, Matrix<float>& out,
int first_col)
{
for (int i = 0;
i < out.nrows();
i++)
{
for (int j = first_col;
j < out.ncols();
j++)
{
out(i, j)
= (float) in[i + (j - first_col)
* out.nrows()];
}
}
}
bool checkClipping(short x)
{
if (x >= SHRT_MAX ||
x <= SHRT_MIN)
{
return true;
} else {
return false;
}
}
};
```

The following is the outline of a source code of the class for which a Recorder class is spun off so that NewDevice can be used. Processing to connect with the device with an initialize function and read data from the device to the () operator is described. Create this source code (NewDeviceRecorder.cpp) in the `librecorder` Folder.

```
#include "NewDeviceRecorder.hpp"
using namespace boost;
NewDeviceRecorder* NewDeviceRecorder::
instance = 0;
// This function is executed in another thread, and
// records acoustic signals into circular buffer.
void NewDeviceRecorder::
operator()()
{
for(;;)
{
// wait during less than (read_buf_size/sampling_rate)
[ms]
usleep(sleep_time);
mutex::
scoped_lock lk(mutex_buffer);
```

```
if (state == PAUSE)
{
continue;
}
else if (state == STOP)
{
break;
}
else if (state == RECORDING)
{
lk.unlock();
// Point5:
Processing to read data from the device is described here.
read_buf = receive_data;
// Point6:
Forward cur_time for the time orresponding to the data read so far.
cur_time += timelength_of_read_data;
mutex::
scoped_lock lk(mutex_buffer);
buffer.insert(buffer.end(), read_buf.begin(), read_buf.begin()
+ buff_len);
lk.unlock();
}
}
}
int NewDeviceRecorder::
Initialize(const string& device_name, int chan_count, int samp_rate, size_t buf_size)
{
// Point7:
Processing to initialize variables and devices to be used is described.
new_device_open_function
}
```

Discussion

The AudioStreamFromMicmodule performs only the processing to read contents of the buffer that the Recorder class has, and the data exchange with devices are performed by each class (ALSARecorder, ASIORecorder, WSRecorder) derived from the Recorder class. Therefore, a new device can be supported by implementing a class corresponding to these derived classes(NewDeviceRecorder).

See Also

The details of how to make a node are described in Section 12.1 "How should I create nodes?".

Chapter 4

Input data generation

4.1 Multi-channel recording

Problem

Read this section when wishing to record multi-channel acoustic signals from a microphone arrays.

Solution

In order to perform multi-channel sound recording, it is necessary to connect to a corresponding device. HARK accepts the RASP series made by System In Fronteir, Inc. and TD-BD-16ADUSB made by Tokyo Electron Device. The network file used for recording is same as the sample in Section 2.1 "First sound recording", Giving proper values to the device type and the device name as parameters enables to perform sound recording. The followings are the parameters when recording sounds for eight channels with RASP.

Table 4.1: Parameter list of AudioStreamFromMic

Parameter name	Type	Default value	Unit	Description
LENGTH	subnetparam	LENGTH	[pt]	FFT length
ADVANCE	subnetparam	ADVANCE	[pt]	Shift length
CHANNEL_COUNT	`int`	8	[ch]	Number of sound recording channel
SAMPLING_RATE	subnetparam	16000	[Hz]	Sampling frequency
DEVICE_TYPE	`string`	RASP		Device type
DEVICE	`string`	/dev/rasp/adc0		Device name

See Also

The details of the sound recording method for one channel are described in Section 2.1 "First sound recording". Moreover, a trouble shooting method for sound recording is described in Section 3.2 "The sound recording cannot be performed well. What should I do?".

4.2 Microphone connection check

Problem

The created system does not work well. It looks like the connection of microphone is doubtful for some reasons. Solution

In order to confirm the connection of microphones, it is necessary to examine both microphone and sound recorder sides. Connect the microphone to another PC and if recording is performed, the sound recorder is the cause. Connect another microphone to the sound recorder and if recording is performed, the microphone is the cause. Check step by step. For details, see "The sound recording cannot be performed well. What should I do? Discussion

When not performed well, it is important to identify (1) to what point the sound recording is performed and (2) from what point it is not. Look for the cause patiently. See Also

The sound recording cannot be performed well. What should I do?

4.3 Sound recording with TED 16ch device

Problem

Wish to use the 16ch device (TD-BD-16AD-USB) made by Tokyo Electron Device for an input to HARK for sound recording.

Solution

TD-BD-16AD-USB is not supported in AudioStreamFromMic. Therefore, in order to use TD-BD-16AD-USB in HARK, it is necessary to install AudioStreamFromMic2, which is a 3rd party module. As for this AudioStreamFromMic2, designating the option (–enable-tdbd16adusb) when configuring enables to use TD-BD-16AD-USB. AudioStreamFromMic2 that corresponds to TD-BD-16AD-USB is created by designating the option and compiling HARK. When using it, set 16 to CHANNEL_COUNT and set SINICH to DEVICE_TYPE and DEVICE.

Table 4.2: Parameter list for using TD-BD-16AD-USB

Parameter name	type	Default value	Unit	Description
LENGTH	`int`	512	[pt]	Frame length as a base unit for processing.
ADVANCE	`int`	160	[pt]	Shift length.
CHANNEL_COUNT	`int`	16	[ch]	Number of sound recording channels
SAMPLING_RATE	`int`	16000	[Hz]	Sampling frequency.
DEVICE_TYPE	`string`	SINICH		Device type.
DEVICE	`string`	SINICH		Device name.

A sample program to record with TD-BD-16AD-USB is included in Record of SampleKit. When executing demoTDBD16ADUSB.sh in this folder, recording is performed for about 1 second and rec0.sw.wav, rec1.sw.wav, ..., rec15.sw.wav are generated. When recording is not performed well, confirm the following items and execute demoTDBD16ADUSB.sh again.

(1) Confirm if microphones are connected.
Confirm if they are unplugged or loosened and connect them properly.

(2) Confirm if sound recording can be performed, using the sample program attached to TD-BD-16AD-USB.
When sound recording is not performed, the OS and driver, or audio interface may not be properly set or connected.

(3) Confirm if AudioStreamFromMic2 is installed in HARK properly.
It is a different module from AudioStreamFromMicand AudioStreamFromMic2. In order to use TD-BD-16AD-USB, AudioStreamFromMic2 needs to be installed.

(4) Execute the network file demoTDBD16ADUSB.n of the samples from a command prompt.
When rec0.sw, rec1.sw, ..., rec15.sw are generated, the sound recording is successfully completed though SoX may not be installed properly.

Discussion

The 3rd party version HARK is equipped with the module (AudioStreamFromMic2), which can record sounds using API attached to TD-BD-16USB. As for this AudioStreamFromMic2, designating –enable-tdbd16adusb when HARK compilation enables to use TD-BD-16AD-USB. Available sampling rates are 42kHz, 32kHz, 16kHz, 8kHz, 44.1kHz, 22kHz and 11kHz.

Inputs-outputs and parameters of the module, and connection with other modules are same as those of AudioStreamFromMic. See Also

The details of the sound recording method for one channel are described in Section 2.1 "First sound recording". Moreover, a trouble shooting method for sound recording is described in Section 3.2 "The sound recording cannot be performed well. What should I do?".

4.4 Impulse response recording

Problem

I wish to measure impulse responses for sound source localization and sound source separation. Solution

To measure impulse responses, prepare a microphone array and speakers to be used. It is necessary to perform the following two items. (1) Sound recording of signals from a speaker and (2) Calculation of impulse responses from the recorded signals.

Here, (2) can be calculated easily with the harktool provided for HARK. **(1) Sound recording of signal**

First, replay TSP (Time Stretched Pulse) signal multiple times and record it. The reason why replaying multiple times is because influences such as reverberation in the room can be suppressed. What is important in the measurement is that sound recording and replaying synchronize. When this cannot be performed, it is necessary to adjust the gap by manual operation, replaying signals only once. In such a case, divide equally the speaker positions on concentric circles around one or two meters away from the microphone array center in order and record sounds. An interval of five degrees is sufficient for recording though it depends on the microphone layouts. **(2) Calculation of impulse response**

Enfold reversed TSP signals to the signals recorded in this way and obtain an impulse response by synchronizing and adding the replay performed multiple times and obtaining mean values. Use harktool for calculation of impulse responses. See the chapter of harktool in the HARK document for details. Discussion

In HARK, impulse responses are used as transfer functions of speech for both sound source localization and sound source separation. Since impulse responses differ depending on positions, it is better to measure as finely as possible. See Also

The chapter of harktool in the hark-document

4.5 Synthesis of 8ch data

Problem

I wish to create data by simulation and perform an operation test offline. Solution

When waveform data of a sound source and an impulse response file are available, multi-channel data can be synthesized.

Synthesis is performed by enfolding impulse responses from a sound source to each microphone in sound source data. High-speed enfolding can be realized by the circulation enfolding method with FFT and so on. Matlab can also be used. Pseudo codes of Matlab are shown below.

```
x=wavread('SampleData1ch.wav');
y1=conv(x,imp1);
y2=conv(x,imp2);
y3=conv(x,imp3);
y4=conv(x,imp4);
```

It is assumed here that 1ch speech is collected in SampleData1ch.wav in Microsoft RIFF format. Moreover, imp1 indicates time expression of an impulse response from the sound source to the microphone 1. In the same way, imp2, imp3 and imp4 indicate time expression of impulse responses from the sound sources to the microphones 2,3 and 4, respectively. y1,y2,y3 and y4 are the multi-channel data synthesized in simulation. Confirm impulse responses and sampling frequency of sound source data before synthesizing. It is necessary to enfold data of the same sampling frequency. For synthesizing sound mixture, add the data enfolded and synthesize them.

Discussion

None. See Also

5.4 Impulse response recording.

4.6 Addition of noise data

Problem

I wish to confirm operation when robot operation noise and fan noise are added to simulation. Solution

Record operation noise or fan noise of the robot with microphones beforehand. After that, the data created by enfolding inpulse responses in clean speech are used. Record each signal waveform in Microsoft RIFF format, give the file names signal.wav and noise.wav. and add and synthesize them. Matlab pseudo codes are shown below.

```
x=wavread('signal.wav');
y=wavread('noise.wav');
z=x+y;
```

z is the simulation acoustic data with noise. Discussion

None. See Also

Noe.

Chapter 5

Acoustic model and language model

5.1 How should I create acoustic model?

Problem

This section describes a construction method of an acoustic model used for speech recognition. It is useful to improve speech recognition performance after having introduced HARK into a robot.

Solution

An acoustic model expresses the relation between a phoneme and acoustic features with a statistical model and is the data that greatly influence the performance of speech recognition. Hidden Markov Model, HMM is usually used often. When changing the microphone layout on the robot or the algorithm and parameters for separation and speech enhancement, it is often the case that the property of acoustic features input into speech recognition changes. Therefore, speech recognition performance is improved by adapting the acoustic model to the condition or newly creating an acoustic model that meets the condition. Here, construction methods for the following three acoustic models (HMM) are described with Hidden Markov Model ToolKit (HTK), which is used when creating an acoustic model of the speech recognition engine Julius used in HARK.

1. Multi-condition training
2. MLLR/MAP adaptation
3. Additional training

Although there actually are various parameters for acoustic models, here, training of the triphone HMM for three states and sixteen mixtures is described as an example. For more information about each parameter, many textbooks are published such as "HTK Book", "IT Text speech recognition system" and others. Please refer to them. A fundamental flow of typical triephone-based acoustic model creation is shown below.

1. Acoustic features extraction
2. Training of monophone model
3. Training of non-context-dependent triephone model
4. Status clustering
5. Training of context-dependent triephone model

Acoustic features extraction

The mel frequency cepstrum coefficient (MFCC) is often used for acoustic features. MFCC can be used though the mel scale logarithmic spectral coefficient (MSLS) is recommended for HARK. MSLS can be created easily from a wav file with a network of HARK. MFCC can also be created with a network of HARK in the same way. However, since MFCC is extracted in HTK, a similar tool HCopy is provided and therefore setting possibility of parameters is higher than that of HARK in terms of MFCC extraction.

```
% HCopy -T 1 -C config.mfcc -S scriptfile.scp
```

```
(Example)
nf001001.dt.wav nf001001.mfc
nf001002.dt.wav nf001002.mfc
...
```

Sample of config.mfcc

```
-----
# HTK Configuration Parameters for Generating MFCC_D_E_N from
# headerless SPEECH Corpus.
# Copyright 1996 Kazuya TAKEDA, takeda@nuee.nagoya-u.ac.jp
# IPA Japanese Dictation Software (1997)
SOURCEFORMAT=NOHEA/D# ASJ Copus has no header part
SOURCEKIND = WAVEFORM
SOURCERATE = 625
# surce sampling frequency is 16 [kHz]
TARGETKIND = MFCC_E_D_Z
TARGETRATE=100000.0 # frame interval is 10 [msec]
SAVECOMPRESSED=F
# set T, if you like to save disk storage
SAVEWITHCRC=F
WINDOWSIZE=250000.0 # window length is 25 [msec]
USEHAMMING=T
# use HAMMING window
PREEMCOEF=0.97
# apply highpass filtering
NUMCHANS=24
# # of filterbank for MFCC is 24
NUMCEPS=12
# # of parameters for MFCC presentation
ZMEANSOURCE=T
# Rather local Parameters
ENORMALISE=F
ESCALE=1.0
TRACE=0
RAWENERGY=F
# CAUTION !!
Do not use following option for nist encoded data.
BYTEORDER=SUN
-----
```

In any event, create an acoustic model with HTK after feature extraction.

Preparation of dictionary and MLF

1. **Data revision:** Generally, even when using a corpus distributed, it is difficult to completely remove fluctuation of description and descriptive errors. Although it is difficult to notice them beforehand, it is preferable to revise them as soon as they are found since such errors cause performance degradations.

2. **Creation of words.mlf:** Create words.mlf for which file names for (virtual) labels corresponding to features and utterance included in the files are written per word. For the words.mlf file, the first line on the file must be #!MLF!#. After describing the label file names with "" from the second line, the utterance included in the label file name is divided for each word and the words are described in their individual lines.

   Moreover, the half size period "." is added to the last line of each entry.

   -exec /phonem/rom2mlf
   ; ¿ words.mlf

3. **Creation of word dictionary:** Create a word dictionary that relates words with phoneme lines. Generally, a dictionary in which word classes are registered is often used. In the case of a small dictionary, it would be enough to describe phoneme lines and words corresponding to them.

   -exec /phonem/rom2dic
   ; — sort — uniq ¿ dic

   -exec /phonem/rom2dic2
   ; — sort — uniq ¿ dic

4. **Creation of phoneme MLF(phones1.mlf):** Create phoneme MLF with a dictionary and word MLF. Use LLEd concretely. Rules are described in phones1.led. Rule to permit sp (short pose) is described in HTKBook. Please see it.

   ```
   % HLEd -d dic -i phones1.mlf phones1.led words.mlf
   ```

   The format of phoneme MLF is almost the same as that of word MLF except that the unit of lines is changed to phoneme from word. An example of phones1.mlf is shown below.

   ```
   ------------------------------
   #!MLF!#

   silB
   a
   r
   a
   y
   u
   r
   u
   g
   e
   N
   j
   i
   ts
   u
   o
   sp

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

Preparation of the list train.scp for features file

Basically, create a file that lists (one file name for one line) feature file names in a complete path. However, since abnormal values may be included in the contents of features files, it is preferable to check the values with HList and list up only normal files.

Preparation of triphone

Although this operation may be performed after trainign of monphone, it may be necessary to remake phones1.mlf depending on the results of the check and therefore perform it here for time saving.

1. Creation of **tri.mlf:** First, simply create a triplicity of phonemes.

   ```
   % HLEd -i tmptri.mlf mktri.led phones1.mlf
   ```

   Phonemes described in mktri.led are removed from the phoneme context.

   ```
   mktri.led
   ------------------
   WB sp
   WB silB
   WB silE
   TC
   ------------------
   ```

   Furthermore, parameters are reduced with short vowel contexts by identifying the anteroposterior long vowel contexts. An example of tri.mlf created is shown below.

   ```
   ------------------------------
   #!MLF!#
   "/hoge/mfcc/can1001/a/a01.lab" silB
   a+r
   a-r+a
   r-a+y
   a-y+u
   y-u+r
   u-r+u
   r-u+g
   u-g+e
   g-e+N
   e-N+j
   N-j+i
   y-i+ts
   i-ts+u
   ts-u+o
   u-o
   sp
   ...
   -------------------------------
   ```

2. **Creation of triphones:** triphones corresponds to the list of triplicities of phonemes included in tri.mlf.

   ```
   grep -v lab tri.mlf |
   grep -v MLF |
   grep -v "\."|
   sort |
   uniq > triphones
   ```

3. **physicalTri:** The triphone list that includes to the phoneme contexts that do not appear in (tri.mlf) at the time of training.

4. **Check of consistency:** Check triphones and physicaiTri. This check is important.

Preparation of monophone

1. **Create prototype (proto) of HMM:** proto can be created with the tool MakeProtoHMMSet in HTK. % ./MakeProtoHMMSet proto.pcf An example of proto.pcf for MFCC is shown below.

```
--------
<BEGINproto_config_file>
<COMMENT>
This PCF produces a 1 stream, single mixture prototype system
<BEGINsys_setup>
hsKind:
P
covKind:
D
nStates:
3
nStreams:
1
sWidths:
25
mixes:
1
parmKind:
MFCC_D_E_N_Z
vecSize:
25
outDir:
./test
hmmList:
protolist/protolist
<ENDsys_setup>
<ENDproto_config_file>
```

Creation of initial model

```
% mkdir hmm0
% HCompV -C config.train -f 0.01 -m -S train.scp -M hmm0 proto
```

As a result, proto and vFloor (initial model) that learned dispersion and means from all the training data are created under hmm0/. Note that it takes time to complete this operation though it depends on data volume.

1. **Creation of initial monophone:**

   - hmm0/hmmdefs Allocate the value of hmm0/proto in all phonemes

     ```
     % cd hmm0
     % ../mkmonophone.pl proto ../monophone1.list > hmmdefs
     ```

monophone1.list is a phoneme list including sp. In HTKBook, "monophone1.list" is to be used after training with the phoneme list of "monophone0.list" without sp. Here, use the phoneme list that includes sp from the beginning.

- hmm0/macros Moreover, create a file "macro" by rewriting some contents of vFloor. This is used as flooring when data are insufficient.

  ```
  % cp vFloor macro
  ```

  In this example, please add the following as a header of macro. Generally, description of a header has to be same as that of hmmdefs (depend on content of proto in other words).

  ```
  -----
  ~o
  <STREAMINFO> 1 25
  <VECSIZE> 25<NULLD><MFCC_E_D_N_Z>
  -----
  ```

```
% cd ../
% mkdir hmm1 hmm2 hmm3
```

Perform training minimum three times repeatedly. (hmm1 hmm2 hmm3) * hmm1

```
% HERest -C config.train -I phones1.mlf -t 250.0 150.0 1000.0 -T 1 \
-S train.scp -H hmm0/macros -H hmm0/hmmdefs -M hmm1
```

* hmm2

```
% HERest -C config.train -I phones1.mlf -t 250.0 150.0 1000.0 -T 1 \
-S train.scp -H hmm1/macros -H hmm1/hmmdefs -M hmm2
```

* hmm3

```
% HERest -C config.train -I phones1.mlf -t 250.0 150.0 1000.0 -T 1 \
-S train.scp -H hmm2/macros -H hmm2/hmmdefs -M hmm3
```

Readjusting alignment settings is to be performed after this. It is omitted here.

Creation of triphone

1. **Creation of triphone by monophone:**

   ```
   % mkdir tri0
   % HHEd -H hmm3/macro -H hmm3/hmmdefs -M tri0 mktri.hed monophones1.list
   ```

2. **Initial training of triphone:**

   ```
   % mkdir tri1
   % HERest -C config.train -I tri.mlf -t 250.0 150.0 1000.0 -T 1 -s stats \
   -S train.scp -H tri0/macro -H tri0/hmmdefs -M tri1 triphones
   ```

   Perform training around 10 times repeatedly.

Clustering

1. **Clustering to the 2000 status:**

```
% mkdir s2000
% mkdir s2000/tri-01-00
% HHEd -H tri10/macro -H tri10/hmmdefs -M s2000/tri-01-00 2000.hed \
triphones > log.s2000
```

Here, 2000.hed is as follows. stats on the first line is an output file obtained in 9.2. Replace thres with a value around 1000 temporally and set it so that the status number becomes 2000 by trial and error, looking at the execution log.

```
----------------------------------------------------------------
RO 100.0 stats
TR 0
QS "L_Nasal" { N-*,n-*,m-* }
QS "R_Nasal" { *+N,*+n,*+m }
QS "L_Bilabial"
{ p-*,b-*,f-*,m-*,w-* }
QS "R_Bilabial"
{ *+p,*+b,*+f,*+m,*+w }
...
TR 2
TB thres "TC_N2_" {("N","*-N+*","N+*","*-N").
state[2]}
TB thres "TC_a2_" {("a","*-a+*","a+*","*-a").
state[2]}
...
TR 1
AU "physicalTri"
ST "Tree,thres"
----------------------------------------------------------------
```

QS Question

TB What is described here is the target of clustering. In this example, only the same central phonemes in the same status are described.

thres Control the final state number by changing the dividing threshold value properly (e.g. 1000 or 1200) (confirm log)

2. **Training:** Perform training after clustering.

```
% mkdir s2000/tri-01-01
% HERest -C config.train -I tri.mlf -t 250.0 150.0 1000.0 -T 1 \
-S train.scp -H s2000/tri-01-00/macro -H s2000/tri-01-00/hmmdefs \
-M s2000/tri-01-01 physicalTri
```

Repeat more than three times

Increase of the number of mixtures

1. **Increase of the number of mixtures (example of 1mix → 2mix):**

```
% cd s2000
% mkdir tri-02-00
% HHEd -H tri-01-03/macro -H tri-01-03/hmmdefs -M tri-02-00 \
tiedmix2.hed physicalTri
```

2. **Training:** Perform training after increase of the number of mixtures.

   ```
   % mkdir tri-02-01
   % HERest -C config.train -I tri.mlf -t 250.0 150.0 1000.0 -T 1 \
   -S train.scp -H s2000/tri-02-00/macro -H s2000/tri-02-00/hmmdefs \
   -M tri-02-01 physicalTri
   ```

   Repeat more than three times Repeat these steps and increase the number of mixtures to around 16 sequentially. Here, it is recommended to double it. (2-¿4-¿8-¿16)

See Also

HTK Speech Recognition Toolkit About acoustic models for Julius Source information of this document Acoustic model construction tutorial for Sphinx developed by CMU Acoustic model construction for Sphinx Acoustic model construction for HTK

5.2 How should I create language model?

Problem

The construction method of language models used for speech recognition is described. Solution

Creation of dictionary for Julis Create a dictionary in the HTK format based on morphologicall-analyzed right text. Use chasen (bamboo tea whisk) for a morphological analysis. Install chasen and create.chasenrc in the Home directory. Designate the directory having grammar.cha in it as "grammar file" and describe the output format as follows.

```
(grammar file /usr/local/chasen-2.02/dic))
(output format "%m+%y+%h/%t/%f\n"))
```

Prepare a right text file and assume this file as seikai.txt. Since it is used for language model creation, insert s , $/s$ in beginning and end of the sentences, respectively.

```
Example of seikai.txt (words do not need to be separated)
<s> Twisted all reality towards themselves. </s>
<s> Gather information in New York for about a week. </s>
:

% chasen seikai.txt > seikai.keitaiso
```

See contents of text.keitaiso, and if there is a wrong part in the morphological analysis, revise it. Moreover, since notion and reading of "he"and "ha" are different, alter their reading to "e", "wa", respectively. In fact, it is necessary to perform normalization of morphemes, removal of unwanted parts other than the above. It is omitted here.

```
Example of seikai.keitaiso
<s>+<s>+17/0/0

+
+75/0/0
</s>+</s>+17/0/0
EOS++
<s>+<s>+17/0/0

% w2s.pl seikai.keitaiso > seikai-k.txt



+
+75/0/0 </s>


% dic.pl seikai.keitaiso kana2phone_rule.ipa |
sort |
uniq > HTKDIC
% gzip HTKDIC
```

```
</s>
[]
silE
<s>
[]
silB

+
+75/0/0
[]
sp


a r a y u r u
```

Termx: Those included in morphological analysis, chasen, HTK format, w2s.pl, dic.pl and kana2phonerule.ipa - vocab2htkdic

Creation of language model for Julis

For creation of language model, see "Speech recognition system" (Ohm sha). However, in order to create 2-gram and reversed 3-gram such as jconf of samples, Sole use of CMU-Cambridge Toolkit is not enough and therefore use palmkit, which is compatible with CMU-Cambridge Toolkit. Moreover, in these days, the reverse 3-gram has become needless for julius and therefore it may be not always necessary to use palmkit. Usage of palmkit is as follows. Prepare correct answer text, assuming it as seikai-k.txt. It is necessary for this file to be finished with a morphological analysis. (In other words, punctuation is regarded as a word and the words are separated with space.) s and $/s$ are inserted to the beginning and end of a sentence, respectively so as to remove transition over s and $/s$. In this case, the description of s and $/s$ is required for the learn.css file.

```
% text2wfreq < learn.txt > learn.wfreq
% wfreq2vocab < learn.wfreq > learn.vocab
% text2idngram -n 2 -vocab learn.vocab < learn.txt > learn.id2gram
% text2idngram -vocab learn.vocab < learn.txt > learn.id3gram
% reverseidngram learn.id3gram learn.revid3gram
% idngram2lm -idngram learn.revid3gram -vocab learn.vocab -context learn.ccs -arpa learn.rev3gram.arpa
% idngram2lm -n 2 -idngram learn.id2gram -vocab learn.vocab -context learn.ccs -arpa learn.2gram.arpa
```

Now, the 2-gram and reverse 3-gram are created. Collect these all together finally. A language model for Julius is created with the tool mkbingram of julius as follows.

```
% mkbingram learn.2gram.arpa learn.rev3gram.arpa julius.bingram
```

See Also

The web page that became a base of this document

Chapter 6

FlowDesigner

6.1 Startup with batch

Problem

FlowDesignerbuilds a module network on GUI and it can be executed on the spot. However, it is troublesome to start GUI at every processing and press the execution button.
Therefore, it is necessary to

- Execute the network created in FlowDesignerby a command line (batch processing)
- Pass variables in the network with a command argument and execute

Solution

The procedure to execute a network of FlowDesignerfrom a command line is as follows:

1. Create a network file (hoge.n) in FlowDesigner.
2. In a command line, enter

   ```
   $ ./hoge.n
   ```

   and execute

The network included in "hoge.n" is executed by FlowDesigner. Moreover, execution conditions can be changed by passing the variables as a command argument without fixing them in the network and the network file can be used universally. Its procedure is as follows:

1. Open the property of a node containing the variable of which the user wishes to pass the argument in FlowDesigner.
2. Designate Type of the variable for which an argument is substituted for in subnet_paramand enter "ARG?" to Value (the index of the argument is to be entered to ?. If it is the first argument, enter "ARG1")
3. Pass the variable as a command argument when executing the network file.
   Example:

   ```
   $ ./hoge.n 0.5 0.9
   ```

By the above procedure, 0.5 of the first argument is passed to ARG1 in "hoge.n" and 0.9 of the second argument is passed to ARG2 in "hoge.n" in the above procedure and the network file is executed. Discussion

The arguments passed in this way are all interpreted as `string`type. When wishing to pass them in `int`type and `float`type, designate such as "`int`:ARG1", "`float`:ARG1". Arguments cannot be passed to `Object`type such as `Vector<int>`by this method. This is a problem due to implementation of FlowDesigner. See Also

None.

Chapter 7

Sound source localization

7.1 How should I perform sound source localization?

Problem

Read this section when

- the user does not understand a parameter setting method for modules used for sound source localization,
- or the user wishes to know an appropriate parameter tuning method under their own use conditions of HARK.

This chapter introduces a parameter tuning method to perform sound source localization efficiently. Solution

Actual processing, visualization and saving in sound source localization are performed with the four modules, which are LocalizeMUSIC, SourceTracker, DisplayLocalizationand SaveSourceLocation.

- **Sound source localization: LocalizeMUSIC**

  1. **AMATRIX**:
     The file that designates A matrix. See How should I create A matrix from the transfer function measured? for the creation method

  2. **NUMSOURCE**:
     The number of sound sources of a localization target. This must be set before performing localization in MUSIC. (See: How many sound sources should I use for LocalizeMUSIC?)

  3. **(MIN | MAX)DEG**:
     Localization target designates a range of an existing direction. When wishing to perform sound source localization for all directions, Set 180 to MAXDEG and -180 to MINDEG. (See: Confirm if successfully localized)

  4. **(LOWER | UPPER)BOUNDFREQUENCY**:
     Set a frequency range used for localization processing in accordance with the property of localization target sound source. For speech, it is usually no problem to use $500 \leq f \leq 2800$, which are the default.

- **Identical sound source tracking: SourceTracker**
  For parameter tuning of SourceTracker, see How should I determine the threshold values for SourceTracker? and How should I determine PAUSE_LENGTH for SourceTracker?

- **Localization result display: DisplayLocalization**
  When wishing to save log of localization result, set `true`to LOGISPROVIDED (default is `false`).
- **Save localization result: SaveSourceLocation**
  For parameter tuning of SaveSourceLocation, see How should I save localization results?

Discussion

See Also

- HARK document: LocalizeMUSIC
- HARK document: SourceTracker
- HARK document: DisplayLocalization
- HARK document: SaveSourceLocation
- How should I improve the system performance?: Sound source localization

7.2 How should I create A matrix from the transfer function measured?

Problem

- Read this section when you do not know a method to create the A matrix from the impulse responses measured.

Solution

Use harktoolfor generation of the A matrix from impulse responses. For usage of harktool, see the HARK document.

Discussion

None. See Also

- HARK document: External tool harktool

7.3 How should I count the number of microphones?

Problem

- Read this section when the user wishes to execute sound source localization designating microphones to be used for processing.
- or wishes to know an influence on sound source localization performance thinning out channel numbers

Solution Adjust SELECTOR parameter of the ChannelSelectormodule. Designate a type of parameter in `Object`and designate only the channel numbers of the microphones to be used for sound source localization in `Vector<int>`. (e.g.: When designate 1st and 3rd third microphones among the four microphones, set to `<Vector<int> 0 2>`.)
Note: Since `int`is set to parameter type for default, make sure to change it to the `Object`type and before designating the number of channels in `Vector<int>` Discussion None. See Also

- HARK document: ChannelSelector

7.4 How many sound sources should I use for LocalizeMUSIC?

Problem

Read this section when

- localization is not performed well when plural sounds sounded simultaneously,
- or the user wishes to know values for the property of the number of sound sources in the LocalizeMUSICmodule.

Solution

For the restriction of methods, the number of sound sources that can be localized simultaneously is equal to or less than the number of microphones to be used. Assuming the number of target sound sources to be localized as N, set this value to N. Furthermore, assuming M noises are heard continuously from a specific direction, set to $M + N$.

Discussion

Theoretically, the MUSIC method can localize sound sources whose number is equal to or less than the number of microphones. In other words, when using eight microphones, the maximum sound sources are eight. However, it has been experimentally confirmed that stable localization can be performed for up to $3 \sim 4$ sound sources.

7.5 Check if the sound source is localized successfully

Problem

Read this section when

- the user wishes to confirm if the sound source localization is successfully performed with the LocalizeMUSICmodule,
- or when performance of processing that uses localization results such as sound source separation with the GHDSSmodule is not good.

Solution

Localize a voice and a sound from a speaker from a specific direction and confirm by collating them with localization results. For confirmation of localization results, use the DisplayLocalizationmodule and SaveSourceLocation. When confirming localization accuracy, set MINDEG of LocalizeMUSICto - 180 and MAXDEG to 180 so that sounds from all directions can be localized. Assuming that a sound does not come from a specific direction, localization results may become stable by giving a restriction to the sound source direction, setting MINDEG, MAXDEG appropriately.

Discussion

The points to improves localization accuracy is to

1. measure transfer functions of the microphone array
2. perform appropriate tuning for the SourceTrackermodule?
3. Set so that localization is not performed from the angle for where there is not a sound source.

See Also

- How should I save localization results in files?
- How should I determine threshold values of SourceTracker?
- HARK document: DisplayLocalizationmodule
- HARK document: SaveSourceLocationmodule

7.6 How should I determine the threshold values for SourceTracker?

Problem

Read this section when

- localization with the LocalizeMUSICmodule is not performed well.
- although there are no sounds, sound sources are continuously localized from a specific direction
- the user wishes to set an appropriate value for the THRESH property of the SourceTrackermodule.

Solution

1. Execute the network file for which the DEBUG property of the LocalizeMUSICmodule is set to `true`.
2. Watch the power value of an MUSIC spectrum during the execution in the case that there are no sounds and that sounds such as clapping are made.
3. Set to a value intermediate between power values of the above two cases.

The point is to set to a value slightly greater than the steady power in the case of silence. Example: If the steady power is around $25.5 - -25.8$ set 26 to THRESH.

Discussion

Since the values that the LocalizeMUSICmodule outputs vary depending on gains of the microphones and surrounding environments, set appropriate to values by trial and error as above. The following trade-off relation arises in setting of THRESH: When set a small value to THRESH, although localization can be performed for small utterance of power, localization of unexpected noise (e.g. sound of footsteps) may be localized. When set a great value to THRESH, although burst sounds around are not reported as a localization result, greater powers are needed for localization of utterance sounds.

See Also

- Check if the sound source is localized successfully
- How should I determine PAUSE_LENGTH for SourceTracker?
- HARK document: LocalizeMUSICmodule
- HARK document: SourceTrackermodule

7.7 How should I determine PAUSE_LENGTH for SourceTracker?

Problem

Read this section when

- although the sound is a continuous sound, localization results are discontinuous.
- all acoustic localization results are output continuously.
- the user wishes to set an appropriate value for the PAUSELENGTH property of the SourceTrackermodule.

Solution

1. Connect the SourceTrackermodule to the DisplayLocalizationmodule and display localization results.
2. Read an appropriate sentence aloud, and see a localization result.
3. **Localization result breaks off:** Raise the value of PAUSE_LENGTH.
4. **Localization result is bound together too much:**

Discussion

The purpose of the PAUSE_LENGTH property is to recognize speech appropriately even if the power of an MUSIC spectrum of the LocalizeMUSICmodule falls by breath of utterance, by localizing it as continuous speech. Since not the sound that breaks off is localized as a continuous sound only limited to utterance of human, such a sound can be used. If your purpose is to localize utterance of human, use the default value.

Unit of PAUSE_LENGTH parameter The unit of this parameter is frame $\times$ 10. Therefore, up to what sec (or msec) the value of PAUSE_LENGTH accepts depends on the sampling frequency designated for the AudioStreamFromMicand AudioStreamFromWavemodules (SAMPLING_RATE) and step size (ADVANCE) of FFT. In the case that all are defaults (sampling frequency: 16000Hz, step size: 160 pt), changing value of PAUSE-LENGTH by 1 corresponds to changing by 1 (msec).

See Also

- How should I use SourceIntervalExtender?
- HARK document: SourceTrackermodule

7.8 How should I use SourceIntervalExtender?

Problem

Read this section when

- the beginning part of the separated sound breaks off.
- the beginning silence section of the separated sound is too long.
- the user does not know how to use the SourceIntervalExtendermodule.

Solution

Adjust as follows.

1. Create the network file, which can save a separation result referring to How should I save separated sounds in files?. In this case, sandwich the SourceTrackerand SourceIntervalExtendermodules between a localization module such as the LocalizeMUSICmodules and a separation module such as GSSor GHDSS.
2. Separate a sound and display or listen to the result
3. **Beginning part of separated sound breaks off:** Raise values of the PREROLLLENGTH property
4. **Beginning silence section of separated sound is too long:** Lower values of the PREROLLLENGTH property

Discussion

At the time point when sound source localization is reported, the time has already elapsed 500 (msec) from the utterance start and the beginning part of the separated sound is lost, which leads to the failure of speech recognition. The role of the SourceIntervalExtendermodule is to solve this problem. Determine the value of PREROLL_LENGTH so as to designate how far to trace back from the start of sound source localization and separate. When decreasing the value of PREROLL_LENGTH too much, the beginning part of a separated sound breaks off, which effects on speech recognition. On the other hand, when increasing the value of PREROLL_LENGTH too much, an utterance is connected to the one before, which may lead to a recognition error in some language models used for speech recognition.

Unit of PREROLLLENGTH The unit of this value corresponds to 1 time frame when performing a Fourier transform for a short time. Therefore, its correspondence with actual time (sec and millisecond) depends on the sampling frequency (SAMPLINGRATE) designated for the AudioStreamFromMicand AudioStreamFromWavemodules and step size (ADVANCE) of FFT. In the case that all are defaults (sampling frequency: 16000Hz, step size: 160 pt), changing value of PREROLL_LENGTH by 1 corresponds to changing by 10 (msec).

See Also

- How should I determine PAUSE_LENGTH of SourceTracker?
- HARK document: The SourceIntervalExtendermodule

7.9 How should I perform two- or three-dimensional localization?

Problem

Read this section when the user wishes to

- to know not only a direction in horizontal surface but also information such as height for the sound coming
- to know the distance to the sound source as well as the sound source direction.

Solution

The current LocalizeMUSICmodule estimates only an incoming direction on a horizontal surface. In order to estimate height of a sound source and distance to the sound source in addition to the above, it is necessary to remodel the program itself of the module. Since the MUSIC algorithm itself does not assume angles of a horizontal surface, it is possible to expand it. However, it will be necessary to have transfer functions for each information to be localized. For example, for estimating height to the coming of sound source, a transfer function from the sound source when changing the horizontal direction and height is required. Moreover, it is necessary to position the microphone array so that required information can be captured. For example, in the case of direction localization of a horizontal surface, position microphones in a circle on a horizontal surface and in the case of estimating height, position them on a ball surface.

Discussion

Since the MUSIC algorithm estimates source locations based on given transfer functions beforehand, it can be applied to height estimation by measuring a transfer function according to the information required. However, since the implementation in HARK is limited to sound source direction estimation of horizontal surfaces, modification is required appropriately.

See Also

- HARK document: LocalizeMUSIC

7.10 How should I save localization results in files?

Problem Read this section when

- the user does not know how to save localization results in files.

Solution Connect the SaveSourceLocationmodule after a module that outputs localization results such as the LocalizeMUSIC, ConstantLocalizationand LoadSourceLocationmodules. Designate a name of a file where localization results are saved in the FILENAME parameter. Discussion None. See Also

- HARK document: SaveSourceLocation

Chapter 8

Separation

8.1 How should I separate sounds?

Problem

This chapter is for when there exist plural sound sources in a microphone input and when the user wishes to separate them to individual sound source waveforms or wishing to separate with better accuracy.

Solution

It is possible to separate sounds of microphone inputs with the sound source separation module GHDSS. See the sound source separation tutorial (tutorial 2) or description of the GHDSSmodule (hark-document) for details. For a tuning method to improve sound source separation performance, see "How should I set step size?", "GHDSS: How should I tune?", "PostFilter: How should I tune?" or "The separation cannot be performed well. What should I do?" Discussion

None. See Also

GHDSS

8.2 How should I save separated sounds in files?

Problem

This section is for when listening to a separated sound for check or saving a separated sound in a file for experiments. Solution

Use the SaveRawPCMmodule basically. Output format is the Raw format of Integer with no headers. Saving methods are different in accordance with values to be saved.

1. Save real number signal (temporal waveform)

When saving temporal waveforms, connect SaveRawPCM. If not connecting a module that performs spectral transform such as MultiFFT, set the value that is same as the dimension number of the input object (INPUT) to the ADVANCE parameter.

2. Save complex signal (spectrum)

When saving spectra, it is necessary to resynthesize them to temporal waveforms first. Therefore, connect the Synthesizemodule and convert spectra into temporal waveforms. After that, connect SaveRawPCM, same as the saving method for real number signals. Here, the ADVANCE parameter must be same as that of the Synthesizemodule.

Judge if it has been saved successfully by seeing if a separated sound file has been generated at the time of execution. Discussion

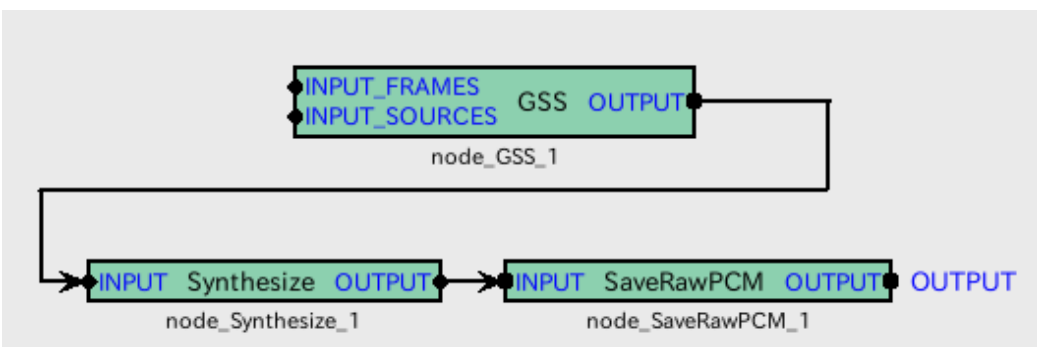


Figure 8.1: Connection example

None

See Also

Synthesize, SaveRawPCM

8.3 GHDSS: How should I create transfer functions?

Problem

I wish to create a transfer function file to be given to the sound source separation module GHDSS. Solution

A transfer function file can be generated by giving sound recording data of TSP signals to harktool. For sound recording of TSP signals, see the recipe "Impulse response recording. For usage of harktool, see description in the HARK document. Discussion

None. See Also

1. Impulse response recording.
2. The chapter of harktool in the HARK document

8.4 GHDSS: How should I designate microphone position?

Problem

This section is for when wishing to separate a sound with the GHDSSmodule with a microphone coordinate, not with impulse responses. Solution

It is necessary to perform the following two steps.

1. Preparation of the file in which microphone coordinate is described: First, it is necessary to prepare the file in which a microphone coordinate is described. See "8.6 Microphone position / noise source" in the hark document to create it.

2. Designate microphone coordinate file in GHDSS: Designate the file in which a microphone coordinate is described in the property window of the GHDSSmodule (the file name is MICPOSFILENAME here). First, set CALC in the GHDSSproperty to Value of the TFCONJ parameter. After that, describe MICPOSFILENAME in Value of the MICFILENAME parameter. When wishing to change setting of the origin of the microphone coordinate system described in MICPOSFILENAME, change the MICPOSSHIFT parameter. When wishing to have the origin at the gravity center of the microphone coordinate, set Value to SHIFT. When using the coordinate described in the file, leave it as FIX. For detailed description, see description of the GHDSSmodule.

Discussion

None. See Also

GHDSS

8.5 GHDSS: How should I designate file of robot noise?

Problem

This section is for when wishing to perform sound source separation considering influences of fixed noise such as fan noise of the robot. The file in which a noise source coordinate of the robot is described is required.

Solution

Perform the following two steps.

1. Creation of noise position file: Basically, describe in the same way as the method for designating a microphone coordinate (How should I designate microphone position?). See "8.6 Microphone position / noise source" in the hark document to create.

2. Designate noise file in GHDSS: First, open the property window of GHDSS. Change the FIXEDNOISE parameter from `false`to `true`. Then, the parameter FIXEDNOISEFILENAME appears. Enter the noise source coordinate file name there. Now, separation is performed without reacting to known noise.

Connect an output module such as SaveRawPCMand check the separated sound to see if the separation has been successfully finished. Separated sounds of the noise source are not output here. Discussion

None. See Also

GHDSS, How should I designate microphone position?

8.6 GHDSS: How should I set step size?

Problem

This section is for when wishing to separate a sound with a fixed beam former or when separation accuracy of GHDSSis not good.

Solution

Try the following methods. **A. Operate for the time being** Open the property of
GHDSS and set Adaptive to Value of the SSMETHOD and LCMETHOD parameters. **B. Separate with fixed beam former**
Set 0 to Value of the SSMETHOD and LCMETHOD parameters and designate an appropriate file in INITWFILENAME that sets an initial value of a separation matrix. Since a file name of this file can be designated in EXPORTW, it is necessary to set noise once and separate it.
Next, Value of each parameter is briefly described. Further, see the item of the GHDSSmodule of in the hark document for details.
Value of the SSMETHOD parameter.
1. FIX
Set FIX to Value of SSMETHOD parameter. Then, the SSMYU parameter appears so enter a value such as 0.001 to Value of it. When this value is great, a separation matrix converges early though the convergence stability and separation accuracy for input data lower. On the other hand, when this value is small, a separation matrix converges slowly though the convergence stability and separation accuracy for input data rise. **2. ADAPTIVE**
Set ADAPTIVE to Value of the SSMETHOD parameter. Since appropriate step size is calculated automatically for input data, a separation matrix converges early and the convergence stability and separation accuracy are high. **3. LCFIX**
Set LCMETHOD to the step size that links. In this case, there are no parameters set in SSMETHOD.
Value of the LCMETHOD parameter.
1. FIX
When setting FIX to Value of the LCMETHOD parameter, the LCMYU parameter appears. Description is the same as SSMETHOD. **2. ADAPTIVE**
When setting LCMETHOD to Value of the LCMETHOD paramete, an optimal value is calculated and set automatically.

Discussion

None. See Also

GHDSS

8.7 GHDSS: How should I set UPDATE_METHOD_TF_CONJ, UPDATE_METHOD_W?

Problem

This section is for when the user does not know how to set UPDATE_METHOD_TF_CONJ and UPDATE_METHOD_W in the property of GHDSS.

Solution

If you do not understand well, use the default value. When changing it, it is necessary to determine if focusing on the sound source itself (ID) or direction (POS). In particular, when a sound source moves or the robot's main body moves, separation results change in the scenes where good separation traceability is needed. When choosing setting to focus on ID, when the same ID is given, the one from the former step is reused as a geometric constraint and separation matrix value. Therefore, when a sound source moves fast, those values become inappropriate as a condition and separation performance falls. On the other hand, when a sound source does not move much, separation accuracy continues improving by reusing. When choosing setting to focus on POS, the characteristic is reversed to the case of ID. It is suited for the case when a sound source moves with a high-speed. This is because when a movement interval is large, a database is used for values of a constraint condition and separation matrix or the system is initialized. An ideal way is to apply these to each sound source dynamically. For details, see description of the GHDSSmodule.

1. UPDATEMETHODTFCONJ
2. UPDATEMETHODW

Discussion

None.

See Also

GHDSS

8.8 GHDSS: How should I tune?

Problem

This section is for when separation performance is not improved well.

Solution

There are parameters that affect separation performance of GHDSS. The separation performance can be improved by setting them properly. The parameters that greatly affect separation performance are as follows.

1. SSMETHOD: Step size parameter
2. TFCONJ: Transfer function parameter
3. INITWFILENAME: Initial value of separation matrix

SSMETHOD Training coefficient of separation matrix. Set it to ADAPTIVE unless special separation is performed.

TFCONJ Designation of a transfer function used for a geometric constraint condition. Since it is used for a constraint condition, separation cannot be performed properly if it is not set to an appropriate value. For details, see description of the GHDSSmodule and harktool.

INITWFILENAME Initial value of separation matrix. It affects accuracy of separation of initial movement. Other affected parts are accuracy of the sound source localization and the number of sound sources, which are inputs of GHDSS. In such a case, tuning of a former module, which mainly is LocalizeMUSICor SourceTracker, is required. Discussion

See Also

"How should I confirm if localization has been successfully finished", The localization cannot be performed well. What should I do?, "How should I set setep size?", "The separation cannot be performed well. What should I do?"

8.9 PostFilter: How should I tune?

Problem

This section is for when distortion is included in the separated sound and when wishing to improve separation performance. Solution

It is necessary to set adequately the parameters of Postfilter for environment. Since the default parameters are determined based on the environment the HARK development team employed, there is no guarantee that they are suited for user's environment. There is a great deal of parameters Postfilter, and many of them are interdependent each other. Therefore, it is extremely difficult to tune by hand operation. One of the solutions is to use the combination optimization method. If a data set is available, apply an optimization method such as Generic Algorithm or Evolutional Strategy by using recognition rates and SNR as evaluation values. Note that the system may learn parameters that are specialized too much for the environment with some data.

Discussion

See Also

"The separation cannot be performed well. What should I do?"

Chapter 9

Feature extraction

9.1 What types of features should I use?

Problem

Read this section when the user does not know what features should be used for speech recognition.

Solution

HARK supports Mel-Frequency Cepstrum Coefficient (MFCC), Mel-Scale Log Spectrum (MSLS)[1] as features for speech recognition. When wishing to perform speech recognition with acoustic models published on the web, use MFCC. When wishing to improve the performance by combining the missing feature theory, use MSLS.

Discussion

The difference between the above two is that whereas MFCC is feature of time domain, MSLS is a feature of frequency domain. When the noise having specific frequency is mixed with acoustic signals, specific features including the frequency are affected for MSLS. On the other hand, in the case of MFCC, an influence of noise spreads and plural features are affected. Therefore, generally, when combine the missing feature theory for speech recognition, MSLS presents good performance. [1] Y. Nishimura, T. Shinozaki, K. Iwano and S. Furui, Noise-robust speech recognition using multi-band spectral features in *Proc. 148th Acoustical Soc. of America Meet.*, 1aSC7 (2004).

9.2 Type of feature

Problem

Read this section when wishing to know what features are available for speech recognition. Solution

The features used for common speech recognition are as follows.

1. LPC(Linear Predictive Coding: Linear prediction) coefficient
2. PARCOR(PARcial CORelated: Partial autocorrelation) coefficient
3. MFCC(Mel-Frequency Cepstrum Coefficient)
4. MSLS(Mel-Scale Log Spectrum)

Discussion

The LPC coefficient is a parameter of a model of spectrum envelope and it is base on the point that the value at the time of t in the stationary process x_t has a correlation with a near past sample. Figure 9.1 shows how to obtain the LPC coefficient. The LPC coefficient is a prediction coefficient (a_m) obtained so that the mean square error of the value ($\hat{x}_t$) predicted from that of M input signals in the past and the value x_t of actual input signals becomes minimum. Since a comparatively precise speech model is obtained in this LPC, it has been used widely for speech analysis-synthesis. However, a model based on LPC has high coefficient sensitivity and it may become unstable due to a slight error of the coefficient. Therefore, in order to deal with this problem, the speech analysis-synthesis is performed in the form of PARCOR. The partial autocorrelation coefficient is a correlation coefficient of prediction errors of x_t (forward prediction) predicted from $x_{t-(m-1)}, \ldots, x_{t-1}$ and x_{t-m} (backward prediction) . Figure 9.2 shows how to derive the partial autocorrelation coefficient. A model based on this PARCOR is stable in principle[1]. MFCC is one of the cepstrum parameters and is a featurederived with filter banks placed at even intervals on a mel frequency axis[1]. Figure 9.3 shows its derivation process. MSLS is derived by a filter bank analysis similar to MFCC, without performing the reverse discrete cosine transform, which is performed at the final step of MFCC extraction processing, and is a feature remaining in the frequency domain.

[1] Hijiri Imai, sound signal processing, Morikita Shuppan Co., Ltd., 1996.
[2] Kiyohiro Shikano et al., IT Text speech recognition system, Ohmsha Co., Ltd., 2001.

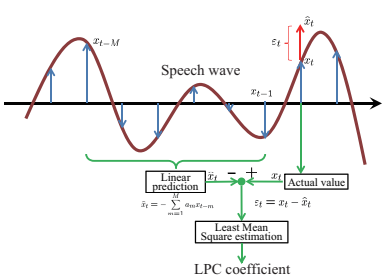


Figure 9.1: LPC coefficients

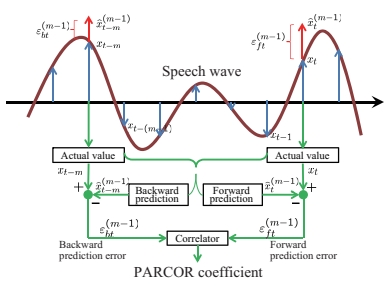


Figure 9.2: PARCOR coefficients

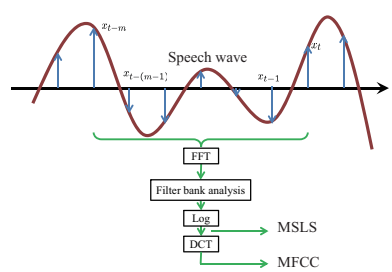


Figure 9.3: MFCC and MSLS

See Also

For MFCC and MSLS, see 10.1 What types of features should I use?.

9.3 How should I save features in files ?

Problem

Read this section when wishing to save features extracted with HARK. Solution

Use the SaveFeaturesmodule to save features. Figure 9.4 shows a network example where features are saved. Here, feature extraction from 1 ch audio signal read from the AudioStreamFromMicmodule is performed and saved. As shown in Figure 9.4, features are saved, assuming the extracted features as inputs of the SaveFeaturesmodule.

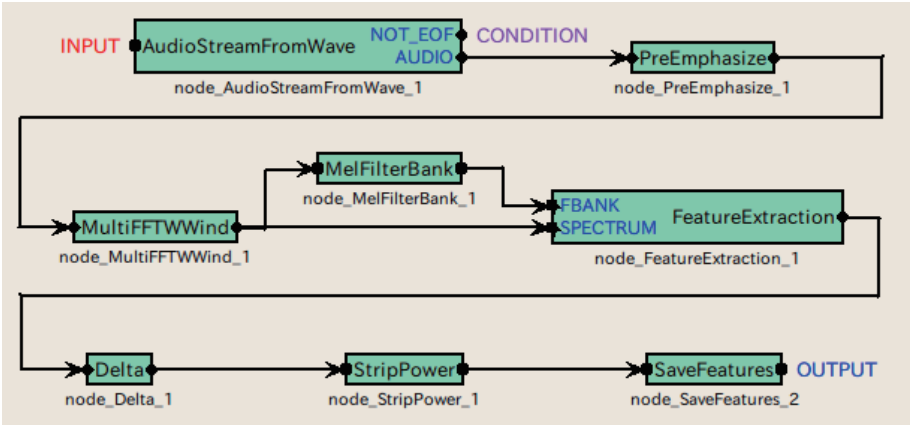


Figure 9.4: Sample netwok to save feature

9.4 How should I set threshold value for MFT?

Problem

Read this section when the user does not know how to set parameters of the MFMGenerationmodule.

Solution

MFMGenerationincludes the parameter THRESHOLD. This parameter affects performance of speech recognition. When setting the threshold value to 0.0, speech recognition is performed not based on the missing feature theory. On the other hand, when setting the threshold value to 1.0, all features are covered with masks and therefore recognition is performed without features at all. In order to obtain a suitable value, it is better to obtain it experimentally through the actual recognition, changing the threshold value by 0.1. Discussion

MFMGenerationis expressed by the following equation, reliability is threshold-processed in THRESHOLD, a mask that uses the two values of 0.0 (unreliable) and 1.0 (reliable) is generated (hard mask).

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

See Also

Chapter 10

Recognition

10.1 .How should I set jconf file?

Problem

This section describes how to designate options of the large vocabulary speech recognition engine Julius when connecting Julius to SpeechRecognitionClient(or SpeechRecognitionSMNClient). Since there are many items for options, options are generally set with .jconf file in a mass.

Solution

Julius includes a lot of settable optional parameters. It is complicated to designate options in Julius every time with command lines. Save a series of optional parameters in a text file. Giving the file to Julius simplifies option inputs. This text file is called .jconf file. Options used in Julius are summarized in http://julius.sourceforge.jp/juliusbook/ja/descop All options can be described in text in .jconf file. Important reminders when connecting Julius to SpeechRecognitionClient(or SpeechRecognitionSMNClient) are summarized as follows.

The setting items required to a minimum are

- -notypecheck
- -plugindir /usr/local/lib/juliusplugin
- -input mfcnet
- -gprune addmasktosafe
- -gram grammar
- -h hmmdefs
- -hlist allTriphones

-notypecheck is an essential option. HARK uses an expanded acoustic parameter structure and it is not supported by type check of the Julius default. Therefore, it is essential to add -notypecheck. When omitting this option, Julius detects a type error in type check of features and does not recognize sounds. For -plugindir, designate the path where function enhancement plug-in files such as mfcnet are saved. This path must be designated before -input mfcnet and -gprune addmasktosafe. All enhancement plugs in files present under the plug-in path are read. -input mfcnet is an option to recognize features received from SpeechRecognitionClient(or SpeechRecognitionSMNClient). Designate this option when wishing to support the missing feature mask. Select the Gaussianpruning algorithm for supporting -gprune. Since calculation is performed with supporting the missing feature mask, the option name is different from normal Julius. Select from $\{add_m ask_t o_s afe||add_m ask_t o_h eu||add_m ask_t o_b eam||add_m ask_t o_n one\}$. They correspond to $\{safe||heuristic||beam||none\}$ of normal Julius, respectively. In addition, it is necessary to

designate a language model and acoustic model, same as normal speech recognition. Designate a grammar file in -gram, a definition file in -h HMM and an HMMList file in -hlist.

Chapter 11

Advanced usage

11.1 How should I create nodes?

Problem

I wish to create a node in HARK by myself but do not understand it only from the material of a HARK training session.

Solution

When creating a new node by the user themselves, it is necessary to it is necessary to compile a source, not a package. For a method to install from a source compilation, see "Installation of HARK" in the HARK training session material. When it is ready, describe a source of the node to be created. For how to make a basic node, the following items are included in "Creation of node" in the HARK training session material.

- Basic form of cc file (source file)
- Description with examples (ChannelSelector)
- Addition of parameter
- Rewriting method of `Makefile.am`

This section further describes the following items showing actual creation of nodes such as `PublisherInt.cc` and `SubscriberInt.cc`

- Addition of input
- Addition of output
- Buffer (`Lookback Lookforward`)
- Input-output of each variable type
- Switching the number of inputs number to dynamic

Creation of PublisherInt.cc First, create `PublisherInt.cc` that reads integers as a parameter and discharge without changing it. (note: the `hark_test` directory is assumed as a package.) Cut and paste the following source code to `{$PACKAGE}/hark_test/src/PublisherInt.cc`.

```
#include <iostream>
#include <BufferedNode.h>
#include <Buffer.h>
using namespace std;
using namespace FD;
class PublisherInt;
DECLARE_NODE(PublisherInt);
/*Node
*
* @name PublisherInt
* @category HARK_TEST
* @description This block outputs the same integer as PARAM1.
*
* @output_name OUTPUT1
* @output_type int
* @output_description This output the same integer as PARAM1.
*
* @parameter_name PARAM1
* @parameter_type int
* @parameter_value 123
* @parameter_description Setting for OUTPUT1
*
END*/
class PublisherInt :
public BufferedNode {
int output1ID;
int output1;
int param1;
public:
PublisherInt(string nodeName, ParameterSet params)
:
BufferedNode(nodeName, params)
{output1ID
= addOutput("OUTPUT1");
output1 = 0;
param1 = dereference_cast<int>(parameters.get("PARAM1"));
inOrder = true;}
void calculate(int output_id, int count, Buffer &out)
{// Main loop routine starts here.
output1 = param1;
cout << "Published :
[" << count << " , " << output1 << "]
" << endl;
(*(outputs[output1ID].
buffer))
[count]
= ObjectRef(Int::
alloc(output1));
// Main loop routine ends here.}};
```

Each part of the source code is described below.

```
#include <iostream>
#include <BufferedNode.h>
#include <Buffer.h>
```

of standard output and library for FlowDesigner. Make sure to include them when creating a node.

```
using namespace std;
using namespace FD;
```

of a name space. Since all the classes of Flowdesigner, which are the base of HARK, are defined in name spaces of FD, make sure to declare when abbrevieting.

```
class PublisherInt;
```

class name of this node. Must be same as the node name set in the following.

```
DECLARE_NODE(PublisherInt);
/*Node
*
* @name PublisherInt
* @category HARK_TEST
* @description This block outputs the same
*
* @output_name OUTPUT1
* @output_type int
* @output_description This output the same
*
* @parameter_name PARAM1
* @parameter_type int
* @parameter_value 123
* @parameter_description Setting for OUTPU
*
END*/
```

declare so that the `PublisherInt` class is defined as one node (an error occurs if not same as the class name). `@name` seen in the comment out below is the setting for the declared node on GUI of Flowdesigner. It is not a comment so make sure to set it. For this set value, four values of "setting of the main body of the node, node inputs, node outputs, node internal parameters" are available. Other than the setting of the main body of the node, plural values can be used (the setting method for plural values are described later). The followings are the concrete set values.

- Setting of the main body of the node
  - `@name`: Node name indicated on Flowdesigner (same as the class name)
  - `@category`: Setting of the category that the node belongs to when right-clicking on GUI of Flowdesigner.
  - `@description`: Description of the node (displayed when mousing on the node in Flowdesigner. Does not have to be input.)
- Setting of inputs of the node
  - `@input_name`: Name of inputs indicated in the node
  - `@input_type`: Type of input variable
  - `@input_description`: Description of the input variable (can be omitted)
- Setting of outputs of the node
  - `@output_name`: Name of outputs indicated in the node
  - `@output_type`: Output variable type
  - `@output_description`: Description of the output variable (can be omitted)
- Setting of internal parameter of the node
  - `@parameter_name`: Variable name indicated in the node (indicated in a yellow window when mouse on)
  - `@parameter_type`: Variable type of the parameter
  - `@parameter_value`: Initial value of the parameter (can be changed in the source)
  - `@parameter_description`: Description of the parameter (can be omitted).

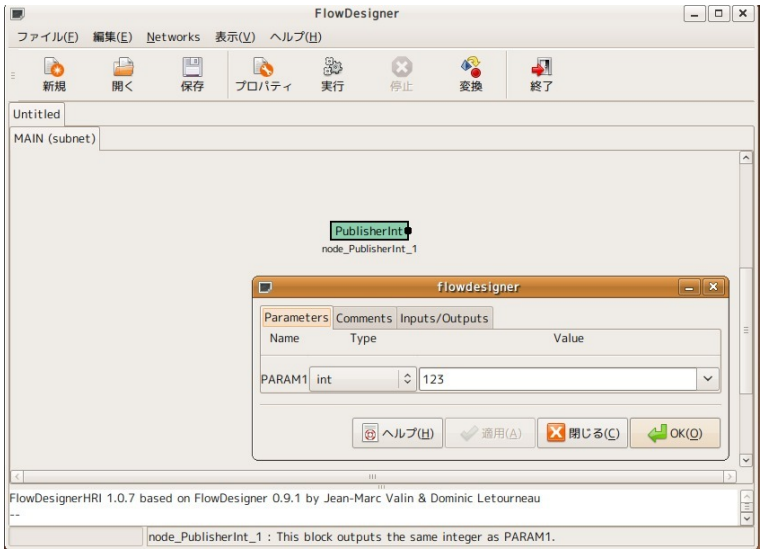


Figure 11.1: PublisherInt node

This source has one output and one internal parameter and therefore they are displayed as follows in Flowdesigner.

```
class PublisherInt :
public BufferedNode {
int output1ID;
int output1;
int param1;
```

Define the `PublisherInt` class that succeeded to `BufferedNode` class. (the `BufferedNode` class is defined in

`outputID` is an integer that stores an ID of an output port. Correspondence is performed for the pointer to be passed to the output port based on this ID,.

```
public: PublisherInt(string nodeName, ParameterSet params) :
BufferedNode(nodeName, params)
{output1ID
= addOutput("OUTPUT1");
output1 = 0;
param1 = dereference_cast<int>(parameters.get("PARAM1"));
inOrder = true;}
```

class. `nodeName` (class object name in N files of Flowdesigner) and `params` (an initializer of the member variable `parameters` of the `Node` class and is a group for which internal parameters are defined for the number of them) are used as arguments. `output1ID = addOutput("OUTPUT1");` becomes a row that stores the ID of `OUTPUT1` set in the GUI side of FlowDesignerin `output1ID` defined in the class side. `param1 = dereference_cast<int>(parameters.get("PARAM1`
is calling the internal parameter set in the GUI side of Flowdesigner casting it into `int`type. There are `int`type, `float`type, `bool`type and `string`type for `@parameter_type` and others are called as Object. (`string`type is called as `Object`and is casted to `string`.) Examples are shown below.

- int type (`int param;`)
  `param = dereference_cast<int>(parameters.get("PARAM"))`
- float type (`float param;`)
  `param = dereference_cast<float>(parameters.get("PARAM"))`
- bool type (`bool param;`)
  `param = dereference_cast<bool>(parameters.get("PARAM"))`

- string type (`string param;`)
  `param = object_cast<String>(parameters.get("PARAM"));`

- Vector type (`Vector<int> param;`)
  `param = object_cast<Vector<int> >(parameters.get("PARAM"));`

The reason why `String` is not `std:: string` and `Vector` is not `std::vector` is because these types are special types for inputs and outputs of Flowdesigner. An error occurs if information is not transferred in this type. When setting `inOrder = true;`, `count` value increases by one every time `calculate` is called (details are described later). Now, back to the description of the source.

```
void calculate(int output_id, int count, Buffer &out)
{// Main loop routine starts here.
output1 = param1;
cout << "Published :
[" << count << " , " << output1 << "]
" << endl;
(*(outputs[output1ID].
buffer))
[count]
= ObjectRef(Int::
alloc(output1));
// Main loop routine ends here.
}
```

is the main routine of the node, the content of this is calculated repeatedly for every `count`. `count`, which is an argument is the loop count. In this node, the value of `PARAM1` is only passed to the next and therefore only the information of a current loop is required. When wishing to calculate an average value over plural loops or to calculate speed, it is necessary to have buffers for the quantity. Details are described later. The value with which `(*(outputs[output1ID]. buffer))[count] = ObjectRef(Int:: alloc(output1));` is output from the node. (The output of a port designated in ID that is called `count` th " `output1ID`" is regulated.) Since this node is has one output, the output can be expressed as `(out[count] = ObjectRef(Int:: alloc(output1));` though it is expressed as above in general types. (In the case of one output, `*(outputs[output1ID]. buffer)` is equivalent to `out`.) Moreover, focus on `output1`, which is `int`type, is casted to `Int`type. In this way, all variable types related to inputs and outputs are to be `Int`type, `Float`type, `String`type, `Bool`type, `Vector`type, `Matrix`type and `Map`type, which are the unique types of Flowdesigner. Examples are shown below.

- `int`type
  `(*(outputs[output1ID].buffer))[count]= ObjectRef(Int::alloc(output1));`

- `float`type
  `(*(outputs[output1ID].buffer))[count]= ObjectRef(Float::alloc(output1));`

- `bool`type
  `(*(outputs[output1ID].buffer))[count]= TrueObject;`

- `string`type
  `(*(outputs[output1ID].buffer))[count]= ObjectRef(new String(output1));`

- `Vector`type
  `RCPtr<Vector<float > > output1(new Vector<float >(rows));`
  `(*(outputs[output1ID].buffer))[count]= output1;`
  (`rows` is the number of elements of the vector. `Vector<int>` can also be defined. include of `Vector.h` is required)

- `Matrix`type
  `RCPtr<Matrix<float > > output1(new Matrix<float >(rows, cols));`
  `(*(outputs[output1ID].buffer))[count]= output1;`
  (`rows`, `cols` are the number of matrixes. `Matrix<int>` can also be defined. include of `Matrix.h` is required)

Here, RCPtr is an object slender pointer for FlowDesigner. This pointer is passed for inputs and outputs of arrays such as Matrixor Vector. Install PublisherInt.cc Compile the source and install it so that PublisherInt.cc

can be used in Flowdesigner. First, add PublisherInt.cc \ an appropriate position in the lib****_la_SOURCES variable of {$PACKAGE}/hark_test/src/Makefile.am (**** is an arbitrary package. hark_test for this example)

Make sure to add "\". In $ cd {$PACKAGE}/hark_test/

```
$ autoreconf;
./configure --prefix=${install dir};
make;
make install;
```

and install. (For $install dir, follow your own setting. /usr/bin corresponds to it.)

Start Flowdesigner. $ flowdesigner

GUI starts, confirm if there is a node careated by Right-click > HARK_TEST> PublishInt

Now the installation is completed. The followings are some trouble shooting for the case that the above are not displayed.

- Confirm if the directory designated in ./configure --prefix=/*** is same as that designated in flow_designer_hri. Since Flowdesigner reads the def file in its own installed directory, it ignores it when it is in other directories.
- Confirm if the script that compiles the own node has been created in {$PACKAGE}/hark_test/src/Makefile. Confirm if autoreconf has been performed properly and if Makefile.am has been rewritten properly.
- Confirm if the node name is same as the class name in the source of the cc file. If it is not same, although it can be compiled, it may not be displayed in GUI.
- Confirm if the path setting is correct (which flowdesiner). This problem arises in the case that the user had ever installed Flowdesigner and HARK in /usr/bin by the root authority in the past and has installed it in the own local by the user authority this time.

Creation of SubscriberInt.cc Create SubscriberInt.cc, which inputs an integer output from PublisherInt.cc

and discharge it without changing it. Cut and paste the following source code in {$PACKAGE}/hark_test/src/SubscriberInt.cc

---

```
#include <iostream>
#include <BufferedNode.h>
#include <Buffer.h>
using namespace std;
using namespace FD;
class SubscriberInt;
DECLARE_NODE(SubscriberInt);
/*Node
*
* @name SubscriberInt
* @category HARK_TEST
* @description This block inputs an integer and outputs the same number with print.
*
* @input_name INPUT1
* @input_type int
* @input_description input for an integer
*
* @output_name OUTPUT1
* @output_type int
* @output_description Same as input
*
END*/
class SubscriberInt :
public BufferedNode {int input1ID;int output1ID;int input1;public:
SubscriberInt(string nodeName, ParameterSet params):
BufferedNode(nodeName, params)
```

```
{input1ID= addInput("INPUT1");
output1ID= addOutput("OUTPUT1");input1 = 0;inOrder = true;}
void calculate(int output_id, int count, Buffer &out)
{// Main loop routine starts here. ObjectRef inputtmp = getInput(input1ID, count);
input1 = dereference_cast<int> (inputtmp);
cout << "Subscribed :
[" << count << " , " << input1 << "]
" << endl;
(*(outputs[output1ID].
buffer))
[count]
= ObjectRef(Int::
alloc(input1));
// Main loop routine ends here.}};
```

Although it is similar to `PublisherInt.cc`, focus on different points.

```
* @input_name INPUT1
* @input_type int
* @input_description input for an integer
```

there is not an input port in `PublisherInt.cc`, when having an input, set GUI of Flowdesigner here first. Format is basically same as that of `@output`. Since `SubscriberInt.cc` has one input instead of not having parameters, the following is indicated in Flowdesigner.

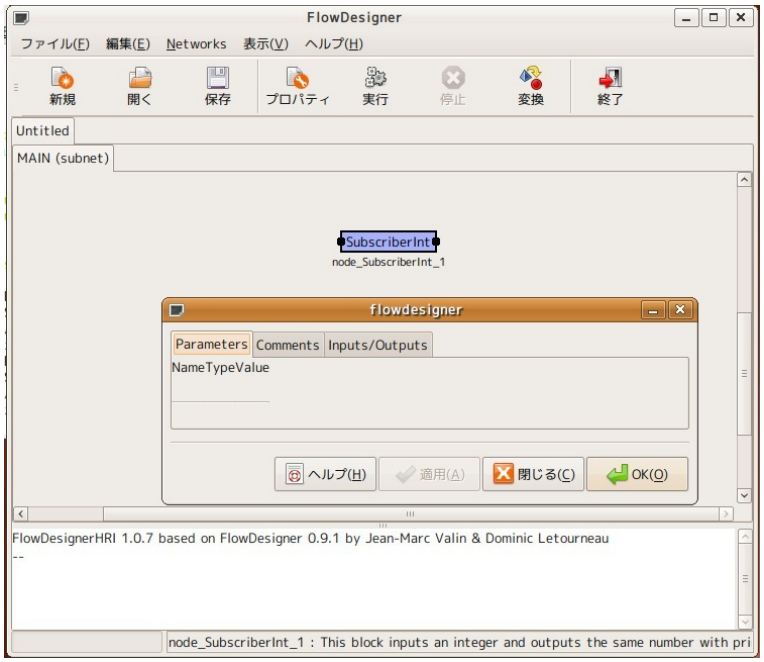


Figure 11.2: SubscriberInt node

```
input1ID= addInput("INPUT1");
```

Register the ID of `INPUT1` set in the GUI side of Flowdesigner in `input1ID` defined in the class side, and correspond the port of GUI to input data be read.

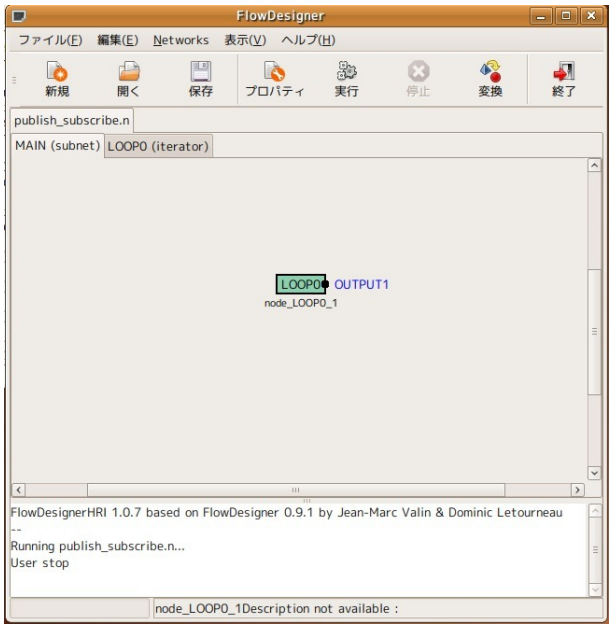
```
ObjectRef inputtmp = getInput(input1ID, count);
input1 = dereference_cast<int> (inputtmp);
```

are received from the input port corresponding to its ID as an original type of Flowdesigner and are casted to `int`type. Like `ObjectRef`(Int type, Float type, Bool type, String type, etc...), which is an original type of FlowDesigneris passed in the output side, recasting is performed in the receiving side in the above process. The followings are examples for other types.

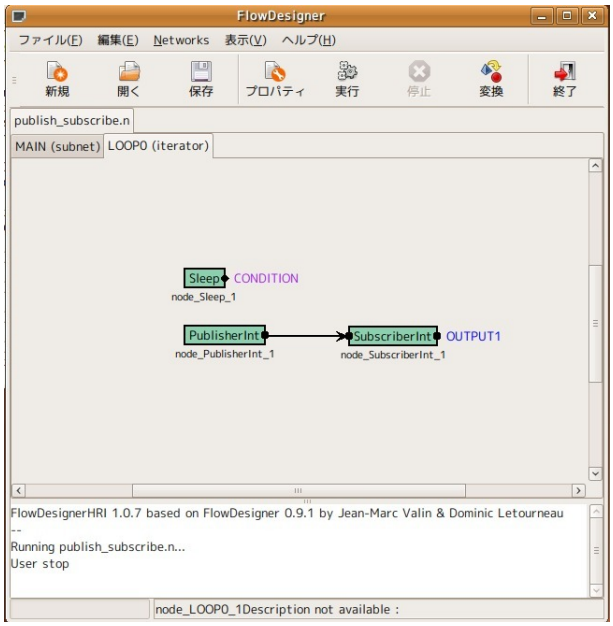
- `int`type
  ```
  ObjectRef inputtmp = getInput(input1ID, count);
  input1 = dereference_cast<int> (inputtmp);
  ```

- floattype
  ```
  ObjectRef inputtmp = getInput(input1ID, count);
  input1 = dereference_cast<float> (inputtmp);
  ```
- booltype
  ```
  ObjectRef inputtmp = getInput(input1ID, count);
  input1 = dereference_cast<bool> (inputtmp);
  ```
- stringtype
  ```
  ObjectRef inputtmp = getInput(input1ID, count);
  const String &input1 = object_cast<String> (inputtmp);
  ```
  (input1 is string type here)
- Vectortype
  ```
  RCPtr<Vector<float > > input1 = getInput(input1ID, count);
  ```
  ((*input1)[i] at the time of use. Same for Vector<int>,)
- Matrixtype
  ```
  RCPtr<Matrix<float > > input1 = getInput(input1ID, count);
  ```
  ((*input1)(i,j)at the time of use. Same for Matrix<int>.)

The power unit is same as PublisherInt.cc. When finishing creation, install by source compilation by the procedure same as that described in the preceding chapter. Start Flowdesigner and confirm that SubscriberInt.cc has been installed properly. Create N file with PublisherInt.cc and SubscriberInt.cc Create a network file (N file) of Flowdesigner using PublisherInt.cc and SubscriberInt.cc finally created. The network file to be created is as follows. If you understand this figure, the rest is not needed so got to the next chapter.



(a) MAIN(subnet) Sheet



(b) LOOP0(iterator) Sheet

Figure 11.3: PublisherInt + SubscriberInt network file

First, start flowdesigner. In the case of a new file, only the sheet of MAIN appears when starting-up. This is the main of the program literally. Networks > Add Iterator

loop processing sheet. (Decide a sheet name properly. LOOP0 here.) First in the MAIN sheet side,

```
Right-click > New Node> Sub
```

that LOOP0 can be executed from the main side. Move to the LOOP0 sheet next and determine sampling period of repetition processing. (either in event base and time base.) Here, perform time- periodical repetition calculation with Sleep.

```
Right-click > New Node> Flow> arranging Sleep node in Sleep
Left-click Sleep > Set parameter \verb|SECONDS| properly (e.g. 10000) > Click the output-termi
```

that the output-terminal of Sleep has become CONDITION. This is a trigger of the loop processing and it means that a new loop begins when the processing of Sleep is completed. The main processing is described next.

```
Right-click> New Node> HARK_TEST> PublisherInt right-click> New Node> HARK_TEST> SubscriberInt
```

LOOP0—Po

two nodes in the LOOP0 sheet.

```
Left-click PublishInt > Set the parameter PARAM1 of PublisherInt properly (e.g.
Connect the output-terminal of PublisherInt to the input of SubscriberInt
Click the output-terminal of SubscriberInt pressing "Shift"
```

that the output-terminal of SubscriberInt has become OUTPUT1. This is the output of the loop processing. One output is created in the LOOP0 block of the MAIN sheet by adding this OUTPUT1 to the output. Back to the MAIN sheet, and set as follows.

```
Click the output-terminal of LOOP0 pressing "Shift"
```

OUTPUT1 comes to be output from the MAIN sheet, which enables to operate proper properly.

```
Name adequately and save
Press "Execute"
```

following output appears in the console.

```
Published :
[0 , 123]
Subscribed :
[0 , 123]
Published :
[1 , 123]
Subscribed :
[1 , 123]
Published :
[2 , 123]
Subscribed :
[2 , 123]
...
```

the network file that can exchange integers has been completed. The above is the basic creation method of nodes. Important applications for creating nodes such as addition of inputs and outputs and processing between plural frames will be described from here.

Add internal parameter to node There is one internal parameter called PARAM1 in PublisherInt.cc. Change PublisherInt.cc like arranging plural parameters here. Since it is not interesting to use inttype, a method to read Vectortype, which is comparatively difficult, as a parameter is described here (named PARAM2.) The goal is to read a Vectortype variable as a parameter, multiply it by verb—PARAM1— times as shown below and alter the node so that results are output from the output port. PARAM1 * PARAM2 Since both addition of an internal parameter in Vectortype and that of an output port in Vectortype are included in this as elements, they are described in two independent chapters. This chapter describes how to add an internal parameter. Open {$PACKAGE}/hark_test/src/PublisherInt.cc. Since it a Vector type variable is treated, include Vector.h first.

```
#include <Vector.h>
```

Change setting of GUI of Flowdesigner. Add the following in /*Node ... END*/.

```
*
* @parameter_name PARAM2
* @parameter_type Vector<int>
* @parameter_value <Vector<int> 0 1 2>
* @parameter_description OUTPUT2 = PARAM1 * P
*
```

see @parameter_value. Format of this <Vector<int> zero 2> is strict. In particular, note that the user must input space and so on. Next, add the followings to the member variables of the class.

```
Vector<int> param2;
```

the following in the constructer.

```
param2 = object_cast<Vector<int> >(parameters.get("PARAM2"));
```

it can be used as Vectoras shown in the following example. `for(int i = 0;i < param2.size();i++){cout << param2[i]<<` this section described the changing part only with sentences A final source is published in the last of the following section.

Add output to node The previous section described how to read variables of Vectortype as internal parameters. In this section, PARAM1 is multiplied by the Vectortype variable and PublisherInt.cc is altered furthermore so that it can be output as Vector type. First, add the output by setting of GUI of FlowDesigneras follows.

```
*
* @output_name OUTPUT2
* @output_type Vector<int >
* @output_description OUTPUT2 = PARAM1 * PARAM2
*
```

Next, add a variable for an ID of the output port to the member variables of the class. `int output2ID;`

the following in the constructer.

```
output2ID
= addOutput("OUTPUT2");
```

the output in the main routine of calculate.

```
RCPtr<Vector<int > > output2(new Vector<int >(param2.size()));
(*(outputs[output2ID].
buffer))
[count]= output2;
```

since it is assumed that calculation of PARAM1 * PARAM2 is performed and results are output, The number of components output becomes same as PARAM2. Therefore, output2 becomes a size of param2.size() pieces. Substitute PARAM1 * PARAM2 for output2 secured. `for(int i = 0;i < param2.size();i++){(*output2)[i]= param1 * param2[i];}` results of the operation of PARAM1 * PARAM2 are output from OUTPUT2. The final revised edition of final PublisherInt.cc is shown below.

```
#include <iostream>
#include <BufferedNode.h>
#include <Buffer.h>
#include <Vector.h>
using namespace std;
using namespace FD;
class PublisherInt;
DECLARE_NODE(PublisherInt);
/*Node
*
* @name PublisherInt
* @category HARK_TEST
* @description This block outputs the same integer as PARAM1.
*
* @output_name OUTPUT1
* @output_type int
* @output_description This output the same integer as PARAM1.
*
* @output_name OUTPUT2
* @output_type Vector<int >
* @output_description OUTPUT2 = PARAM1 * PARAM2
*
* @parameter_name PARAM1
* @parameter_type int
* @parameter_value 123
* @parameter_description Setting for OUTPUT1
*
* @parameter_name PARAM2
* @parameter_type Vector<int>
* @parameter_value <Vector<int> 0 1 2>
* @parameter_description OUTPUT2 = PARAM1 * PARAM2
*
END*/
class PublisherInt :
public BufferedNode {
int output1ID;
int output2ID;
int output1;
int param1;
Vector<int> param2;
public:
```

```
PublisherInt(string nodeName, ParameterSet params)
:
BufferedNode(nodeName, params)
{output1ID= addOutput("OUTPUT1");output2ID= addOutput("OUTPUT2");output1 = 0;param1 = dereference_cast<int>(parameters.get("PARAM1"
void calculate(int output_id, int count, Buffer &out)
{// Main loop routine starts here.
output1 = param1;
cout << "Published :
[" << count << " , " << output1 << "]
" << endl;
(*(outputs[output1ID].
buffer))
[count]= ObjectRef(Int::alloc(output1));RCPtr<Vector<int > > output2(new Vector<int >(param2.size()));
(*(outputs[output2ID].
buffer))
[count]= output2;
cout << "Vector Published :
[";for(int i = 0;i < param2.size();i++){(*output2)[i]= param1 * param2[i];cout << (*output2)[i]<< " ";}
cout << "]
" << endl;
// Main loop routine ends here.}};
```

Add input to node A variable of Vector type was output to the new `PublisherInt.cc` above. This chapter describes hot to create new `SubscriberInt.cc` that receives a Vector type variable as an input. Open `{$PACKAGE}/hark_test/src/SubscriberInt.cc` by an appropriate editor. Since a Vector type variable is treated, include `Vector.h` first.

```
#include <Vector.h>
```

setting of GUI of Flowdesigner. Add the following in `/*Node ... END*/`.

```
*
* @input_name INPUT2
* @input_type Vector<int >
* @input_description input for a Vector
*
```

a member variable of the class.

```
int input2ID;
```

the following in the constructer.

```
input2ID= addInput("INPUT2");
```

input data in the main routine of `calculate`.

```
RCPtr<Vector<int > > input2 = getInput(input2ID, count);
```

`input2` can be used in the main routine freely as follows.

```
for(int i = 0;i < (*input2).
size();i++){cout << (*input2)[i]<< " ";}
```

final revised edition of `SubscriberInt.cc` is shown below.

```
#include <iostream>
#include <BufferedNode.h>
#include <Buffer.h>
#include <Vector.h>
using namespace std;
using namespace FD;
class SubscriberTutorial;
DECLARE_NODE(SubscriberTutorial);
/*Node
*
* @name SubscriberTutorial
* @category HARKD:Tutorial
* @description This block inputs an integer and outputs the same number with print.
*
* @input_name INPUT1
* @input_type int
* @input_description input for an integer
*
* @input_name INPUT2
* @input_type Vector<int >
* @input_description input for a Vector
*
* @output_name OUTPUT1
* @output_type int
* @output_description Same as input
*
END*/
class SubscriberTutorial :
```

```
public BufferedNode {
int input1ID;
int input2ID;
int output1ID;
int input1;
public:
SubscriberTutorial(string nodeName, ParameterSet params)
:
BufferedNode(nodeName, params)
{input1ID= addInput("INPUT1");input2ID= addInput("INPUT2");output1ID= addOutput("OUTPUT1");input1 = 0;inOrder = true;}
void calculate(int output_id, int count, Buffer &out)
{// Main loop routine starts here.
ObjectRef inputtmp = getInput(input1ID, count);
input1 = dereference_cast<int> (inputtmp);
cout << "Subscribed :
[" << count << " , " << input1 << "]
" << endl;
(*(outputs[output1ID].
buffer))
[count]
= ObjectRef(Int::
alloc(input1));
RCPtr<Vector<int > > input2 = getInput(input2ID, count);
cout << "Vector Received :
[";for(int i = 0;i < (*input2).size();i++){cout << (*input2)[i]<< " ";}
cout << "]
" << endl;
// Main loop routine ends here.}};
```

Create N file with new PublisherInt.cc and SubscriberInt.cc The fundamental procedure is same as that for N file described in the previous section. First, confirm if compilation / installation are performed properly. When successfully installed, start flowdesigner. Open the N file created in the previous section. Confirm that `OUTPUT2` has been add to the output part of the `PublisherInt` node and `INPUT2` has been added to the input part of the `SubsctiberInt` node. (If you cannot confirm it, the installation has not been successfully completed so see trouble shooting in the previous section) Connect them. Moreover, the following setting is necessary.

`Left-click PublishInt > Set type of PARAM2 of PublisherInt to object > OK` Save and press "Execute".

```
Published :
[0 , 123]
Vector Published :
[0 123 246 ]
Subscribed :
[0 , 123]
Vector Received :
[0 123 246 ]
Published :
[1 , 123]
Vector Published :
[0 123 246 ]
Subscribed :
[1 , 123]
Vector Received :
[0 123 246 ]
Published :
[2 , 123]
Vector Published :
[0 123 246 ]
Subscribed :
[2 , 123]
Vector Received :
[0 123 246 ]
...
```

above is indicated in the console, confirm `Vector`is exchanged correctly. The following is the capture of the node created.

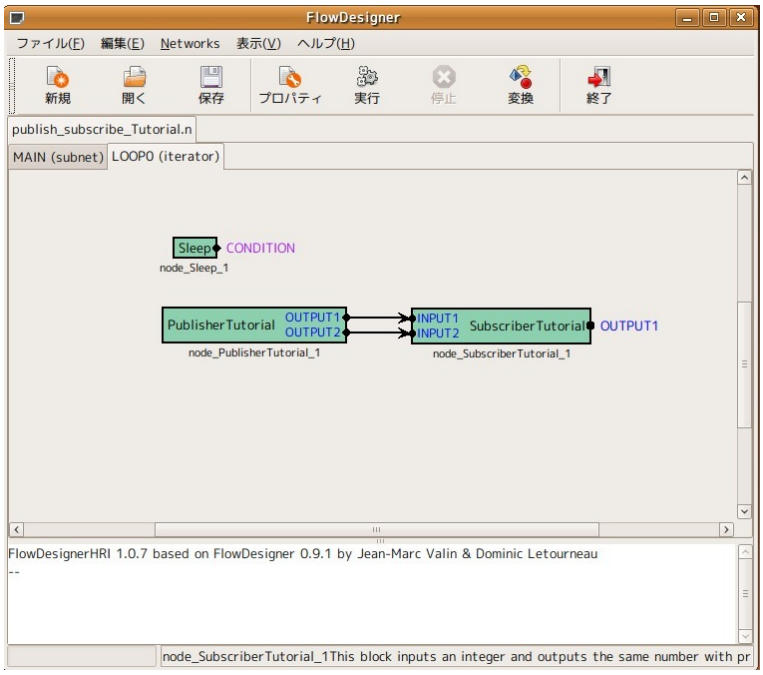


Figure 11.4: SubscriberInt + PublisherInt network file

Processing over plural frames (lookBack, lookAhead option) Until now, the calculation that can be processed only with the concerned frame has been treated, such as multiplying vectors by constant numbers. However, when calculating a mean of frames and differentiating, processing over frames is required. This chapter describes reminders for such a case. For example, create a block to obtain the total of two frames before and after the current frame, five frames in total.

$$OUTPUT1(t) = INPUT1(t-2) + INPUT1(t-1) + INPUT1(t) + INPUT1(t+1) + INPUT1(t+2)| \quad (11.1)$$

SInce `getInput` is maintained for each of the ID and `count` value of the input port, it is calculated as follows.

```
total = 0.0;
for(int i = -2;i <= 2;i++){ObjectRef inputtmp = getInput(input1ID, count + i);input1 = dereference_cast<in
```

Here, the contents of `getInput` is expressed as "`count + i`" and therefore the processing can be performed for the two frames before and after the current frame. However, an error occurs in this processing. The first reason is that `count` value demands -2 and -1 in the first and second frames, which causes the error "It is not present". This problem is solved simply by clipping with the following.

```
if(count >= 2)
```

The second reason is that in Flowdesigner, unless particularly requested, only one frames before and the after the current frame are maintained in its specification and therefore when trying to see frames more than two frames away, which causes the error "It is not present". Describing the following in the constructer enables to maintain information for the number of frames in the buffer.

```
inputsCache[input1ID].
lookAhead = 2;
inputsCache[input1ID].
lookBack = 2;
```

above is a declaration to secure buffers in the two frames before and after the current frame. Set adequately for your own calculation. Now, the calculation can be performed for between frames. The followings are the `int`type inputs and the "`AccumulationInt`" node that acquires the total of frames for between `SUM_BACKWARD` frames before and `SUM_FORWARD` frames after. Since the above `lookAhead` and `lookBack` can be changed as internal

parameters, note that an error occurs if not secured sufficiently. A node construction example of Flowdesigner is shown below.

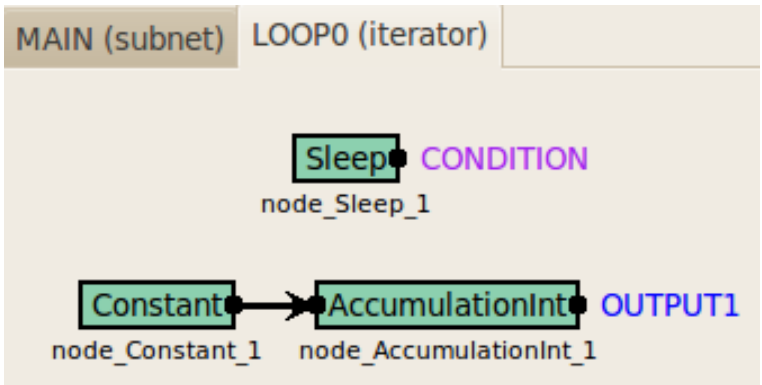


Figure 11.5: AccumulationInt network file

```
#include <iostream>
#include <BufferedNode.h>
#include <Buffer.h>
using namespace std;
using namespace FD;
class AccumulationInt;
DECLARE_NODE(AccumulationInt);
/*Node
*
* @name AccumulationInt
* @category HARKD:Tutorial
* @description This block takes a summation over several frames of the input.
*
* @input_name INPUT1
* @input_type int
* @input_description input for an integer
*
* @output_name OUTPUT1
* @output_type int
* @output_description total
*
* @parameter_name SUM_FORWARD
* @parameter_type int
* @parameter_value 5
* @parameter_description Forward buffer for summation
*
* @parameter_name SUM_BACKWARD
* @parameter_type int
* @parameter_value -5
* @parameter_description Backward buffer for summation
*
* @parameter_name LOOK_FORWARD
* @parameter_type int
* @parameter_value 0
* @parameter_description Forward buffer for summation
*
* @parameter_name LOOK_BACKWARD
* @parameter_type int
* @parameter_value 0
* @parameter_description Backward buffer for summation
*
* @parameter_name IN_ORDER
* @parameter_type bool
* @parameter_value true
* @parameter_description inOrder setting
*
END*/
class AccumulationInt :
public BufferedNode {
int input1ID;
int output1ID;
int input1;
int sum_forward;
int sum_backward;
int look_forward;
```

```
int look_backward;
int total;
bool in_order;
public:
AccumulationInt(string nodeName, ParameterSet params)
:
BufferedNode(nodeName, params)
{input1ID= addInput("INPUT1");output1ID= addOutput("OUTPUT1");input1 = 0;
sum_forward = dereference_cast<int>(parameters.get("SUM_FORWARD"));
sum_backward = dereference_cast<int>(parameters.get("SUM_BACKWARD"));
look_forward = dereference_cast<int>(parameters.get("LOOK_FORWARD"));
look_backward = dereference_cast<int>(parameters.get("LOOK_BACKWARD"));
inputsCache[input1ID].
lookAhead = look_forward;
inputsCache[input1ID].
lookBack = look_backward;
in_order = dereference_cast<bool> (parameters.get("IN_ORDER"));
inOrder = in_order;}
void calculate(int output_id, int count, Buffer &out)
{total = 0;if(count + sum_backward >= 0){for(int i = sum_backward;i <= sum_forward;i++){ObjectRef inputtmp = getInput(input1ID, cou
cout << "AccumulationInt :
[" << count << " , " << input1 << " , " << total << "]
" << endl;
(*(outputs[output1ID].
buffer))
[count]
= ObjectRef(Int::
alloc(total));}};
```

Processing that does not calculate every frame (inOrder option) In contrast to the previous section, this chapter describes processing that is performed once in several frames. For example, consider the following source.

```
if(count % 3 == 0){
ObjectRef inputtmp = getInput(input1ID, count);
input1 = dereference_cast<int> (inputtmp);}
```

In this case, `getInput` is read not for not every `count`, but once in three times. In Flowdesigner, the former node is processed in accordance with the request from the latter `getInput` as a specification and a feature. In other words, in the case of a source like the above, the node request inputs once in three times and therefore its former node is calculated as one processing for three. The following is an example. First, as a preparation, create the following `SubscriberIntWithPeriod` node.

```
#include <iostream>
#include <BufferedNode.h>
#include <Buffer.h>
using namespace std;
using namespace FD;
class SubscriberIntWithPeriod;
DECLARE_NODE(SubscriberIntWithPeriod);
/*Node
*
* @name SubscriberIntWithPeriod
* @category HARKD:Tutorial
* @description This block inputs an integer and outputs the same number with print with specific period.
*
* @input_name INPUT1
* @input_type int
* @input_description input for an integer
*
* @output_name OUTPUT1
* @output_type int
* @output_description Same as input
*
* @parameter_name PERIOD
* @parameter_type int
* @parameter_value 1
* @parameter_description Period of INPUT1 subscription
*
* @parameter_name IN_ORDER
* @parameter_type bool
* @parameter_value true
* @parameter_description inOrder setting
*
```

```
END*/
class SubscriberIntWithPeriod :
public BufferedNode {
int input1ID;
int output1ID;
int input1;
int period;
bool in_order;
public:
SubscriberIntWithPeriod(string nodeName, ParameterSet params)
:
BufferedNode(nodeName, params)
{input1ID= addInput("INPUT1");output1ID= addOutput("OUTPUT1");input1 = 0;period = dereference_cast<int>(parameters.get("PERIOD"));i
void calculate(int output_id, int count, Buffer &out)
{// Main loop routine starts here.
if(count % period == 0){ObjectRef inputtmp = getInput(input1ID, count);input1 = dereference_cast<int> (inputtmp);}
cout << "Subscribed :
[" << count << " , " << input1 << "]
" << endl;
(*(outputs[output1ID].
buffer))
[count]
= ObjectRef(Int::
alloc(input1));
// Main loop routine ends here.}};
```

In the `SubscriberIntWithPeriod` node, `getInput` is performed simply by the cycle set in `PERIOD`, and its value is displayed. Next, create the `CountOutput` node to output the current `count` value without changing it.

```
#include <iostream>
#include <BufferedNode.h>
#include <Buffer.h>
using namespace std;
using namespace FD;
class CountOutput;
DECLARE_NODE(CountOutput);
/*Node
*
* @name CountOutput
* @category HARKD:Tutorial
* @description This block outputs the count number
*
* @output_name OUTPUT1
* @output_type int
* @output_description This output the same integer as PARAM1.
*
* @parameter_name IN_ORDER
* @parameter_type bool
* @parameter_value true
* @parameter_description inOrder setting
*
END*/
class CountOutput :
public BufferedNode {int output1ID;bool in_order;public:CountOutput(string nodeName, ParameterSet params):BufferedNode(nodeName, pa
{output1ID= addOutput("OUTPUT1");in_order = dereference_cast<bool> (parameters.get("IN_ORDER"));inOrder = in_order;}
void calculate(int output_id, int count, Buffer &out)
{cout << "CountOut :
[" << count << "]
" << endl;
(*(outputs[output1ID].
buffer))
[count]
= ObjectRef(Int::
alloc(count));}};
```

When the two nodes are completed, build a network file of Flowdesigner as follows.

First, set both the `CountOutput` node and `IN_ORDER`, which is an internal parameter of the `SubscriberIntWithPeriod` node, to "`false`". (this the default value of Flowdesigner.) With `PERIOD` of `SubscriberIntWithPeriod` as 3, exe-

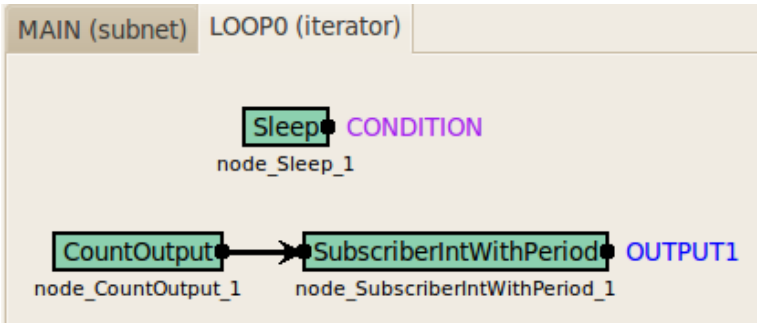


Figure 11.6: SubscriberIntWithPeriod network file

cute network file. Then the output to the console is as follows.

```
CountOut :
[0]
Subscribed :
[0 , 0]
Subscribed :
[1 , 0]
Subscribed :
[2 , 0]
CountOut :
[3]
Subscribed :
[3 , 3]
Subscribed :
[4 , 3]
Subscribed :
[5 , 3]
CountOut :
[6]
Subscribed :
[6 , 6]
Subscribed :
[7 , 6]
Subscribed :
[8 , 6]
CountOut :
[9]
Subscribed :
[9 , 9]
...
```

this way, since the latter `SubscriberIntWithPeriod` request inputs once in three times, `CountOut` is processed only once in three times. It makes it difficult to perform print debug When there is processing such as differentiation or count calculation in `CountOut`, process cannot be performed correctly. (When wishing to confirm this problem, implement a counter that counts every time `calculate` is actually called by the `CountOut` node. It is confirmed that the counter does not work for all `count`.) The "`inOrder`" option solves this problem. Set `IN_ORDER` to `true` in the aforementioned network file. The following result is obtained when executing it.

```
CountOut :
[0]
Subscribed :
[0 , 0]
Subscribed :
[1 , 0]
Subscribed :
[2 , 0]
CountOut :
[1]
CountOut :
[2]
CountOut :
[3]
Subscribed :
[3 , 3]
Subscribed :
[4 , 3]
Subscribed :
[5 , 3]
CountOut :
[4]
CountOut :
[5]
CountOut :
[6]
Subscribed :
[6 , 6]
Subscribed :
[7 , 6]
Subscribed :
[8 , 6]
CountOut :
[7]
CountOut :
[8]
CountOut :
[9]
Subscribed :
[9 , 9]
...
```

In this way, the loop processing is executed for `CountOut` three times properly in accordance with the calling that is performed once in three times. When wishing to calculate `count` times properly for all nodes regardless of the calling, Make sure to enter the following to the constructor. `inOrder = true;`

```
#include <iostream>
#include <BufferedNode.h>
#include <Buffer.h>
using namespace std;
using namespace FD;
class SummationInt;
DECLARE_NODE(SummationInt);
/*Node
*
* @name SummationInt
* @category HARKD:Tutorial
* @description This block outputs INPUT1 + INPUT2 + INTPUT3
*
```

```
* @input_name INPUT1
* @input_type int
* @input_description input for an integer
*
* @input_name INPUT2
* @input_type int
* @input_description input for an integer
*
* @input_name INPUT3
* @input_type int
* @input_description input for an integer
*
* @output_name OUTPUT1
* @output_type int
* @output_description Same as input
*
END*/
class SummationInt :
public BufferedNode {
int input1ID;
int input2ID;
int input3ID;
int output1ID;
int input1;
int input2;
int input3;
int total;
public:
SummationInt(string nodeName, ParameterSet params)
:
BufferedNode(nodeName, params)
{input1ID= addInput("INPUT1");input2ID= addInput("INPUT2");input3ID= addInput("INPUT3");output1ID = addOutput("OUTPUT1");inOrder =
void calculate(int output_id, int count, Buffer &out)
{// Main loop routine starts here.
input1 = 0;
input2 = 0;
input3 = 0;
ObjectRef inputtmp1 = getInput(input1ID, count);
input1 = dereference_cast<int> (inputtmp1);
ObjectRef inputtmp2 = getInput(input2ID, count);
input2 = dereference_cast<int> (inputtmp2);
ObjectRef inputtmp3 = getInput(input3ID, count);
input3 = dereference_cast<int> (inputtmp3);
total = input1 + input2 + input3;
cout << "SummationInt :
[" << count << " , " << total << "]
" << endl;
(*(outputs[output1ID].
buffer))
[count]
= ObjectRef(Int::
alloc(total));
// Main loop routine ends here.}};
```

This node can be created easily by the method that has been described so far. When the node is created, create a network file as follows.

In this case, 1+1+1, so it is natural that 3 is output. Now, here, suppose that you wish to perform addition for two variables from this node not for three variables. In this case, it looks like the calculation can be performed by opening one input port and giving integers to two ports as shown in the following figure. However, when executing, unfortunately, an error occurs because information is not all in the input port.

In Flowdesigner, except special cases, even if the data are not used for processing inside the node, all information about the input ports read in `getInput` must be received in the former step as a specification. In lowers multi-usability and it is troublesome to prepare all inputs for each type in some cases Therefore, there is a method to change the number of input ports to dynamic with the member function of the `Node` class, which is a parent class of the `BufferedNode` class of `translateInput`. Now, create a node for which the number of input ports is variable with the same example.

```
#include <iostream>
#include <BufferedNode.h>
#include <Buffer.h>
using namespace std;
using namespace FD;
class SummationInt;
DECLARE_NODE(SummationInt);
```

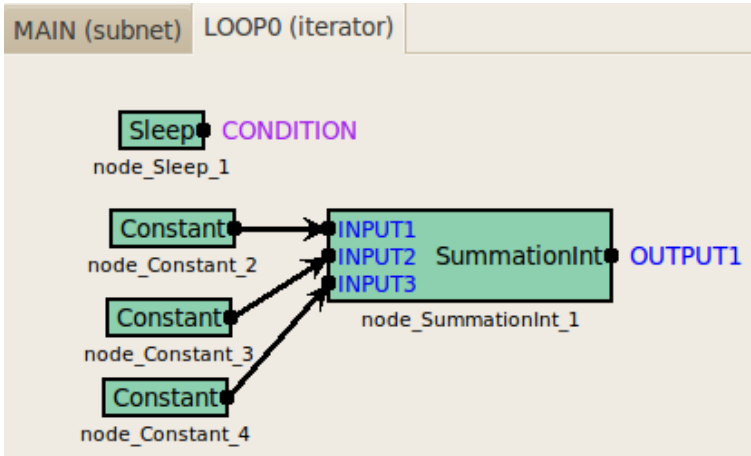


Figure 11.7: SummationInt network file

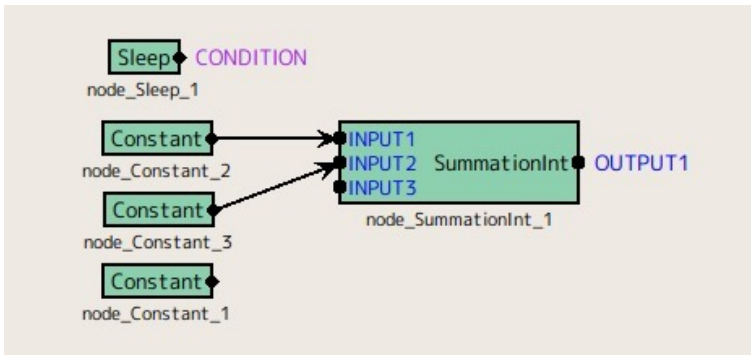


Figure 11.8: SummationInt network file (input port imperfection)

```
/*Node
*
* @name SummationInt
* @category HARKD:Tutorial
* @description This block outputs INPUT1 + INPUT2 + INTPUT3
*
* @input_name INPUT1
* @input_type int
* @input_description input for an integer
*
* @input_name INPUT2
* @input_type int
* @input_description input for an integer
*
* @input_name INPUT3
* @input_type int
* @input_description input for an integer
*
* @output_name OUTPUT1
* @output_type int
* @output_description Same as input
*
END*/
class SummationInt :
public BufferedNode {
int input1ID;
int input2ID;
int input3ID;
int output1ID;
int input1;
int input2;
int input3;
int total;
public:
```

```
SummationInt(string nodeName, ParameterSet params)
:
BufferedNode(nodeName, params), input1ID(-1), input2ID(-1), input3ID(-1)
{output1ID= addOutput("OUTPUT1");inOrder = true;}
virtual int translateInput(string inputName)
{if (inputName == "INPUT1"){return input1ID = addInput(inputName);}
else if (inputName == "INPUT2")
{return input2ID = addInput(inputName);}
else if (inputName == "INPUT3")
{return input3ID = addInput(inputName);}
else {throw new NodeException(this, inputName+ " is not supported.", __FILE__, __LINE__);}}
void calculate(int output_id, int count, Buffer &out)
{// Main loop routine starts here.
input1 = 0;
input2 = 0;
input3 = 0;
if (input1ID != -1){
ObjectRef inputtmp1 = getInput(input1ID, count);
input1 = dereference_cast<int> (inputtmp1);}
if (input2ID != -1){
ObjectRef inputtmp2 = getInput(input2ID, count);
input2 = dereference_cast<int> (inputtmp2);}
if (input3ID != -1){ObjectRef inputtmp3 = getInput(input3ID, count);
input3 = dereference_cast<int> (inputtmp3);}
total = input1 + input2 + input3;
cout << "SummationInt :
[" << count << " , " << total << "]
" << endl;
(*(outputs[output1ID].
buffer))
[count]
= ObjectRef(Int::
alloc(total));
// Main loop routine ends here.}};
```

The change points are as follows.

- The input port ID is initialized with -1 in the constructor.
  By doing it this way, name so that the ID becomes -1 when the input port is opened.
- Added `translateInput`.
  Only the inputs for which this function is connected for the number of input ports are called, showing which port is connected, as an argument. If the input port is connected, an ID of the true input port is acquired.
- Processing is performed so that inputs are read in the `calculate` function when the ID is other than -1.
  Now, processing for the cases where the input is opened or unopened can be performed.

Compiling this source and using it in Flowdesigner, you will see that the calculation is performed properly without depending on the number of inputs. See Also

- Installation from source compilation
  See "Installation of HARK" of the HARK short course material
- How to make basic node
  See "Creation of node" of the HARK short course material

11.2 How should I improve the system performance?: Processing speed

Problem

How should I raise the computing speed of HARK?

Solution

Processing speed of HARK is dominated basically by complexity of nodes that the user built and algorithms of the nodes that the user created. As examples, the more you perform processing of eigenvalue expansion in LocalizeMUSIC, processing of save of SaveFeaturesor indication by `cout` and `cerr`, the processing time required for one count gets longer. Simple node construction that meets the purpose is the best for speeding up. (The algorithms have been improved for the nodes presently used in HARK in consideration of real time performance.) For construction of simple node, the following two methods are useful to improve processing speed, though it would be slight improvement.

1) `Comment-in IN_ORDER_NODE_SPEEDUP`
   This function calls in the end of the node class. (Comment-in what are described concretely in cc files such as LocalizeMUSIC)

2) Change the optimization option for compiling.
   Realize by adding the option when making. Optimization is performed with a stronger condition in order of -O, -O1, -O2, -O3 and the processing speeds up in accordance with it.
   Concretely, in the case of -O2,
   `/src/Makefile.am`
   Add the following to the above

```
libhark_d_la_CXXFLAGS = @GTK_CFLAGS@ -O2
CXXFLAGS = -g -O2
CFLAGS = -g -O2
FFLAGS = -g -O2
```

Discussion

Evaluate the performance of each as follows. As evaluation, the patterns for which compiled with options of -O, -O1, -O2, -O3, and those for which `IN_ORDER_NODE_SPEEDUP` is further added, eight patterns in total, are compared for processing time. For the algorithm used for comparison, simple processing below is performed for each node and processing time is measured in one hundred of the nodes connected in series.

```
int count_time = 100000000;
for (i = 0;i < count_time;i++)n = n + i;
```

Table 11.9 shows a result of the case without `IN_ORDER_NODE_SPEEDUP` and Table 11.10 shows that with it. As shown in the results, significant differences are not seen in computer time, the processing speeds up by 3 percent by a combination of optimization option and `IN_ORDER_NODE_SPEEDUP`.

See Also

None

Figure 11.9: Comaparison of Elapsed Time without INORDERNODESPEEDUP

Option	O3	O2	O1	O
	14.2408	12.7574	14.0147	14.1765
	13.9518	14.0789	14.2417	14.3901
	13.912	14.0633	14.5486	13.7121
	14.3929	13.9978	14.2038	14.1017
	13.7976	14.3931	13.8478	14.2374
	14.0315	13.9962	14.5201	14.1924
	14.3108	14.0069	14.1044	14.1694
	14.0055	14.3397	14.2014	14.5729
	14.004	14.0419	14.467	14.1911
	14.4457	13.8734	14.1159	14.2177
Total	141.0926	139.5486	142.2654	141.9613
Average	**14.10926**	**13.95486**	**14.22654**	**14.19613**

Figure 11.10: Comaparison of Elapsed Time with INORDERNODESPEEDUP

Option	O3 + speedup	O2 + speedup	O1 + speedup	O + speedup
	14.0007	13.8055	14.3469	14.4444
	14.3702	13.5448	13.9894	14.1628
	14.0753	14.371	14.4229	13.8679
	12.9333	13.8942	14.1801	14.5209
	14.398	13.8926	13.7115	14.0369
	13.6696	14.1745	14.5278	14.7882
	14.0837	14.0613	13.9905	14.5343
	14.4443	14.018	14.0915	14.1182
	13.0798	14.4962	14.4936	14.5952
	13.6339	14.1081	14.1904	14.2751
Total	138.6888	140.3662	141.9446	143.3439
Average	**13.86888**	**14.03662**	**14.19446**	**14.33439**

11.3 How should I improve the system performance?: Sound source localization

Problem

How should I adjust the parameters when performance of sound source localization is bad?

Solution

Solutions are described for each problem.

Q.1) **Localization directions are indicated sloppily or are not indicated at all.**

When displaying a localization result in DisplayLocalization, localization directions are indicated sloppily in some cases. This is because even low power source has been localized as a sound source. When results are not indicated at all, it is because of the opposite reason.

A.1-1) **Change `THRESH` of SourceTracker**

A.1-2) **Make `NUM_SOURCE` of LocalizeMUSICequal to the number of target sounds**

(1-1) is the parameter that directly changes the threshold value of an expectation value of a sound source direction and therefore adjust appropriately so that a peak of only the sound source is captured well. (1-2) enhances the peak of the target sound direction with NULL space and therefore the number of peaks to be enhanced changes according to setting of `NUM_SOURCE` (number of sound sources). When setting this wrong, the performance deteriorates such as that a peak rises in a noise direction or it does not appear for the target direction (In actual localization, dull peaks, which still can be used, are obtained in most cases.) In the case that there is one speaker, the performance will be improved by setting 1 to it.

Q. 2) **Only one peak appears even though there are plural sound sources**

A.2-1) **Change** `MIN_SRC_INTERVAL` **in SourceTracker**

A.2-2) **Make** `NUM_SOURCE` **of LocalizeMUSICequal to the number of the target sounds**

As for (2-1), when there is a sound source near by, for example, for two sound sources whose incoming directions are only 10 degrees away from each other, it is necessary to set a value less than 10 degrees to `MIN_SRC_INTERVAL`. When the set value is greater than this, the two sound sources are localized as one sound source.

Q.3) **Other problems**

A.3-1) `LOWER_BOUND_FREQUENCY` **of LocalizeMUSIC,** `UPPER_BOUND_FREQUENCY`
In the sound source localization, a method for which correlation matrixes are added and their mean is calculated for the frequencies designated in these two and therefore If setting a frequency that is totally different from that of the target frequency, a dull peak is obtained from averaging. Use frequencies that correspond to those of sound sources.

A.3-2) `MIN_DEG` **of LocalizeMUSIC,** `MAX_DEG`
Sound source localization is performed only for the range determined by designating these two values. When wishing to perform localization for 360 degrees, make sure to designate 180 degrees and -180 degrees.

See Also

For details of the algorithms for sound source localization, see "Technical description of HARK (sound source localization) " of the HARK short course material. Moreover, for detail description of the parameters, see "Tutorial1: Construction of sound source localization system of the HARK short course material

11.4 How should I improve the system performance?: Sound source separation

Problem

How should I adjust the parameters when performance of sound source separation is bad?

Solution

This section describes setting of the following nodes GHDSS, PostFilterand MFMGenerationfor sound source separation.

1) **Main body of sound source separation (GHDSS)**

1-1) **Setting matching known spatial transfer function**

Since GHDSSperforms separation with information of spatial transfer functions measured beforehand (or with microphone positions), setting in accordance with the concerned transfer functions is required. Concretely, they correspond to the following set values.

- `LC_CONST = DIAG`
  When transfer functions are measured properly it is set to `FULL`, though it is set to `DIAG` in most cases.
- **Setting of** `TF_CONJ`
  `TF_CONJ = DATABASE`: Transfer function mode actually measured
  Designate a transfer function file in `TF_CONJ_FILENAME`
  `TF_CONJ = CALC`: Mode to calculate from conventional position information
  Designate microphone layout in `MIC_FILENAME`

1-2) **Designation of curvature of non-linear constraint**

In GHDSS, the coefficient `SS_SCAL`, which corresponds to the curvature (gradient the origin) of a sigmoidal function determines the performance. It becomes closer to the linear constraint by raising this curvature and the non-linearity is raised by lowering it. Since too much lowering would result in dull adaptation, perform setting that matches the condition by changing this value.

1-3) **Measure to stationary noise**

- Separation under the assumption of stationary noise with `FIXED_NOISE` = `true`

1-4] **Step size**

There are two kinds of step sizes, which are `SS_METHOD` and `LC_METHOD` (one is for cost error and the other is for step size of linear constraint). Performance improved by setting `ADAPTIVE` to both, unless the environment is extremely optimized.

2) **Post filter**

2-1) **PostFilter**

For this processing, better recognition performance is obtained without PostFilter, depending on the situation. In PostFilter, stationary noise, reverberation and noise leakage are dynamically estimated by basically using magnitude relations of input signal power and highly precise separated sounds can be obtained by subtracting them. Performance falling in some conditions because distortion of speech in such subtraction hurts recognition. Therefore, since the magnitude of the influence of PostFiltercan be changed by varying the influence caused by the three estimation of stationary noise, reverberation and noise leakage, the performance can be improved depending on conditions. The influence of PostFilteris minimized by setting 0 to the following parameters.

- Leakage: `LEAK_FACTOR = 0`
- Reverberation: `REVERB_LEVEL = 0`
- Stationary noise: `LEAK_FACTOR = 0`

When wishing to raise the influence of PostFilter, make these values closer to 1.

2-2) **WhiteNoiseAdderafter PostFilter**

Adjust the value of `WN_LEVEL`. When it is too small, the distortion generated in the separated sound cannot be loosened sufficiently. When too large, not only a distortion part but also separated sound itself is affected.

3) **Missing Feature Mask**

3-1) **MFMGeneration**

Threshold values to mask features can be changed by changing the value of `THRESHOLD` in the range from 0 to 1 . All features are not masked and treated as 0 as it gets close to 1 and all features are masked and treated as 1 as it gets close to 0 so the recognition rate falls.

Discussion

See above description of Solution

See Also

For details of the algorithms for sound source separation, see "Technical description of HARK (sound source separation)" of HARK the short course material.

11.5 How should I create debug module?

Problem

How should I create a module to debug the own created system?

Solution

Basically

- display information of the inside in the print statement and check.

Confirm a message on the console by describing the followings.

- `cout << message1 << endl;`
- `cerr << message2 << endl;`

Furthermore, when executing the own created network file (assumed as `nfile.n` here) as below on a command line, not from GUI of flowdesigner,

- `. /nfile.n> log.txt`

  the message (above `message1`) discharged in `cout` is saved in `log.txt`. and when executing it as below,

- `./nfile.n 2> log.txt`
  the message (above `message2`) discharged in `cerr` is saved in `log.txt`.

There is a parameter named `DEBUG` in blocks such as LocalizeMUSIC. Messages are discharged only when it is set to `true`. As a useful function, only the messages that the user wishes to confirm are discharged by attaching such a switch.

11.6 How should I use debug tool?

Problem

Are there any methods to debug other than creating a module?

Solution

The following three items are the main candidates.

1) The debug that uses gdb in a command line

2) Error confirmation of transfer function by HARKtool

For HARKtool, it is to improve localization and separation performance rather than to debug the program. Brief descriptions are given for each item below.

1) **Debug that uses gdb in command line**

gdb is a free source level debugger of GNU and is a useful tool that can stop execution by designating a break point or perform execution for each line. It is used as follows for debugging operation in HARK.

- Compile with a `-g -ggdb` option
  (Add `AM_CXXFLAGS = -g -ggdb` to `Makefile.am`)
- `gdb /path/flowdesigner`
  Move to the gdb console here.
  `(gdb)`
- Setting of break pointer
  `(gdb) b MODULE::calculate` (when designating with function name)
  `(gdb) b module.cc: 62` (when designating with line count. the 62nd line of `module.c`)
- Confirmation of break pointer
  `(gdb) i b`
- Break point with condition (example)
  `(gdb) cond 3 i==500` (Stop the third break point in the loop processing after executing 500 times)
- Normal execution
  `(gdb) r nfile.n` (Execute `nfile.n`)
- Stepwise execution (execute for each line)
  `(gdb) n`
- Restart execution (when stopping at the break point)
  `c nfile.n`

2) **Error confirmation of transfer function by HARKtool**

For (2), see description in the section of harktool in this manual.

See Also

For how to create a debug module that which uses break point with gdb, see "Implementation: Modification of Channel Selector" of the HARK short course material

11.7 How should I connect with other systems by TCP/IP?

Problem

I have understood that sound source localization and sound source separation can be performed in HARK. How should I use the information for other systems? For this question, the followings are described separately.

- Connection between other systems created in HARK
- Connection with other systems that use socket communication

Solution 1

: Connection between different systems created for HARK In many cases, users may wish process a collection of plural submodules as one network with a system created for HARK as one submodule. For example, when performing parallel computation for the same algorithms, it is time consuming to describe all parallel module groups to one subnet sheet and it would be fairly complicated since a large number of node have to be treated. This section describes the method to treat one system described in the subnet sheet as one node.

1) **Describe the system to be connected**
   You may add a subnet sheet to the existing n file as
   `"Networks" -> "Add Network"`
   , or you may use the existing n file. In this case, make sure to **designate inputs and outputs** for this sheet (it is not MUST since inputs and outputs can be changed after connecting).

2) **Export the created subnet sheet (Note 1)**
   Under the condition that the created sheet is active,
   `"Networks" -> "Export Network"`
   give an appropriate name and save a n file only for the created subnet sheet. In this case, since the subnet sheet name is maintained when it is imported later, it is recommended to give a simple name before exporting.

3) **Import in the file to be connected**
   In the file the created subnet sheet is used as one node,
   `"Networks" -> "Import Network"`
   and select the exported file. The file is read as a new subnet sheet.

4) **Use as submodule**
   Open the sheet different from the imported subnet sheet and prepare a sheet used as a submodule. Right click the new sheet and select the name same as that of the own created subnet sheet from
   `"New Node" -> "Subnet"`
   and the node with the number of inputs and outputs and name that are same as those of the imported subnet sheet can be used.

5) **Change of submodule (option)**
   The imported subnet sheet can be changed as wished. See (Note 2) for examples of this change.

Sub-modularization of this large system save time to create the same block construct and makes it easy to see the large-scale network file. The following cases are considered as usage examples of this function.

A) **Use the same processing multiple times**
   When repeating the same processing or performing parallel computation, if the processing is modularized, the user may only need to use the node repeatedly for the main sheet. As an extreme example, it is easily imaginable that when repeating the same processing 100 times continuously, the configuration becomes simple by calling the module 10 times with the sheet for 10 times of processing as a submodule, rather than connecting 100 times.

B) **Use the processing always used as template**
For example, in the case that the same block configuration is always used for sound source localization parts and sound source separation parts are always changed, it would be easier to export the processing once as a template and read it as one node every time, rather than to describe processing of sound source localization every time expressly.

(Note 1) Even if not exporting and importing expressly, it is alright to copy and paste the node of the subnet sheet to be exported to the importing side. However, it is troublesome to copy and paste all the nodes of such a large subnet sheet, it is recommended to export.

(Note 2) Indicate below about change example of a submodule.

5-1) **Change of the number of inputs and outputs and name**
This may be a matter of course. The number of inputs and outputs and the names can be changed after importing so that the specification of the side to be used as a node can be changed. Since the number of inputs and outputs and name of the main sheet to be used as a node can be changed by changing the number of inputs and outputs and the name, it is necessary to build a network accordingly again not to break the causal sequence of the connection.

5-2) **Change parameters in submodule one by one**
It is often the case that when using the imported submodule multiple times, the user wishes to give an argument so that the parameter set in the subnet sheet can be changed for each calling. In such a case, use `"subnet_param"`, which is a parameter type. The usage is simple. Take the argument in the imported subnet sheet which imported and give appropriate names to the values, assuming types of the parameters to be changed as `"subnet_param"` for each. Then, parameters with the names are added to the blocks called as nodes. Now, give values to the arguments in the calling side.

Solution 2

: Connection with other systems used for socket communication This chapter describes about connection with systems except HARK for which socket communication is used. Concretely, the current HARK realizes connection with Julius, which is a speech recognition engine, by socket communication. As a node for communication, the followings things are in HARK by default.

1) SpeechRecognitionClient

2) SpeechRecognitionSMNClient

Either clients send information on vocal features vector as messages through a socket and the processing is performed in the Julius side. For details, see the above two sources. However, since the above nodes perform a little complicated processing such as sending one speech section in a mass The following section describes simple methods to make a HARK node as a client and as a server.

Solution 2-1 : Connection with the other systems for which socket communication is used (client) In the case of communication with Julius, Julius listens to a message from HARK as a server program continuously. Build this example by creating a simple server program and HARK node. First create the server program `listener.c`. (this corresponds to another system other than HARK in the Julius side.)

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
main(int argc, char *argv[])
{inti;intfd1, fd2;
struct sockaddr_in
saddr;
struct sockaddr_in
caddr;
int
len;
int
ret;
char buf[1024];
if (argc != 2){
printf("Usage:
listener PORT_NUMBER\n");
exit(1);}
if ((fd1 = socket(AF_INET, SOCK_STREAM, 0))
< 0)
{perror("socket");
exit(1);}
bzero((char *)&saddr, sizeof(saddr));
saddr.sin_family = AF_INET;
saddr.sin_addr.s_addr = INADDR_ANY;
saddr.sin_port = htons(atoi(argv[1]));
if (bind(fd1, (struct sockaddr *)&saddr, sizeof(saddr))
< 0){
perror("bind");
exit(1);}
if (listen(fd1, 1)< 0)
{perror("listen");exit(1);}
len = sizeof(caddr);
if ((fd2 = accept(fd1, (struct sockaddr *)&caddr, &len))< 0)
{perror("accept");exit(1);}
close(fd1);
ret = read(fd2, buf, 1024);
while (strcmp(buf, "quit\n")!= 0)
{printf("Received Message :
%s", buf);
ret = read(fd2, buf, 1024);
write(fd2, buf, 1024);
// This returns a responce.}
close(fd2);}
```

The contents are program of normal socket communication.

Perform `printf` display of a message written in at `fd2` by this program. After cutting and pasting a source, compile it. `$ gcc -o listener listener.c` I create a client node of the HARK side. Cut and paste the following source to create `TalkerTutorial.cc`.

```
#include <iostream>
#include <BufferedNode.h>
#include <Buffer.h>
#include <string.h>
#include <sstream>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <csignal>
using namespace std;
using namespace FD;
class TalkerTutorial;
DECLARE_NODE(TalkerTutorial);
/*Node
*
* @name TalkerTutorial
* @category HARKD:Tutorial
* @description This block outputs the same integer as PARAM1 and sends it through socket.
*
* @output_name OUTPUT1
* @output_type int
* @output_description This output the same integer as PARAM1.
*
* @parameter_name PARAM1
```

```
* @parameter_type int
* @parameter_value 123
* @parameter_description Setting for OUTPUT1
*
* @parameter_name PORT
* @parameter_type int
* @parameter_value 8765
* @parameter_description Port number for socket connection
*
* @parameter_name IP_ADDR
* @parameter_type string
* @parameter_value 127.0.0.1
* @parameter_description IP address for socket connection
*
END*/
bool exit_flag2 = false;
class TalkerTutorial :
public BufferedNode {
int output1ID;
int param1;
int port;
string ip_addr;
struct sockaddr_in
addr;
struct hostent *hp;
int
fd;
int
ret;
public:
TalkerTutorial(string nodeName, ParameterSet params)
:
BufferedNode(nodeName, params)
{output1ID= addOutput("OUTPUT1");param1 =dereference_cast<int>(parameters.get("PARAM1"));port= dereference_cast<int>(parameters.get
ip_addr = object_cast<String>(parameters.get("IP_ADDR"));
signal(SIGINT, signal_handler);
signal(SIGHUP, signal_handler);
signal(SIGPIPE, signal_handler);
if ((fd = socket(AF_INET, SOCK_STREAM, 0))
< 0)
{perror("socket");exit(1);}
bzero((char *)&addr, sizeof(addr));
if ((hp = gethostbyname(ip_addr.c_str()))== NULL)
{perror("No such host");exit(1);}
bcopy(hp->h_addr, &addr.sin_addr, hp->h_length);
addr.sin_family = AF_INET;
addr.sin_port = htons(port);
if (connect(fd, (struct sockaddr *)&addr, sizeof(addr))
< 0){perror("connect");exit(1);}
inOrder = true;}
void calculate(int output_id, int count, Buffer &out)
{// Main loop routine starts here.
ostringstream message;
message << "[" << count << " , " << param1 << "]
" << endl;
string buf = message.str();
write(fd, buf.c_str(), 1024);
cout << "Sent Message :
[" << count << " , " << param1 << "]
" << endl;
(*(outputs[output1ID].
buffer))
[count]
= ObjectRef(Int::
alloc(param1));
if(exit_flag2){
cerr << "Operation closed...
" << endl;
close(fd);
exit(1);}
// Main loop routine ends here.}
static void signal_handler(int s)
{exit_flag2 = true;}};
```

The content of this is also a client node of simple socket communication. Socket communication is performed for character strings stored in `message` for every `count` loop. After cutting and pasting, perform installation from the source compilation. When a server and a client are ready, build a node of Flowdesigner. It is a simple node that performs socket communication for every specific time cycle in `Sleep` as shown below.

In order to execute, start the server program first. Decide an appropriate port number (8765 here) and perform as follows on a command line.

```
$ ./listener 8765
```

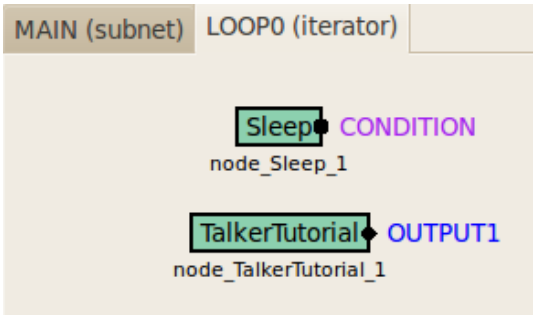


Figure 11.11: Network file: TalkerTutorial

move to setting of the network file of Flowdesigner. Start a new console, and start the network file created in Flowdesigner above.

```
Left click the Sleep node > Set an appropriate cycle to SECONDS in float type (10000 he
Left click the TalkerTutorial node > Set an appropriate to PARAM1 in int type
Left click the TalkerTutorial node > Set the port number to PORT in int type (8765 here
Left-click the TalkerTutorial node > Set an IP address or a host name to IP_ADDR in str
```

the setting of IP address, it is assumed here that a server and a client work with the same machine. Of course it is possible to communicate with a remote machine.

```
Click "Execute" button
```

messages from HARK comes to the operating server and they are indicated in each concole as follows. Server side (listener)

```
Received Message :
[0 , 123]
Received Message :
[1 , 123]
Received Message :
[2 , 123]
...
```

Cli side (Flowdesigner: TalkerTutorial)

```
Sent Message :
[0 , 123]
Sent Message :
[1 , 123]
Sent Message :
[2 , 123]
...
```

this way, responses are given to a system other than HARK by socket communication. Solution 2-1 : Con-

nection with the other systems for which socket communication is used (server) Next, create a server node that can receive data from some client program by socket communication. Same as above, after preparing a simple client program, create a server node. First create the server program `Talker.c`. It is a simple client program of socket communication.

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
main(int argc, char *argv[])
{struct sockaddr_in
addr;
struct hostent *hp;
int
fd;
int
len;
int
```

```
port;
char buf[1024];// intret;
if (argc != 3){
printf("Usage:
talker SERVER_NAME PORT_NUMBER\n");
exit(1);}
if ((fd = socket(AF_INET, SOCK_STREAM, 0))
< 0)
{perror("socket");exit(1);}
bzero((char *)&addr, sizeof(addr));
if ((hp = gethostbyname(argv[1]))
== NULL)
{perror("No such host");exit(1);}
bcopy(hp->h_addr, &addr.sin_addr, hp->h_length);
addr.sin_family = AF_INET;
addr.sin_port = htons(atoi(argv[2]));
if (connect(fd, (struct sockaddr *)&addr, sizeof(addr))
< 0){
perror("connect");
exit(1);}
while (fgets(buf, 1024, stdin))
{write(fd, buf, 1024);
// ret = read(fd, buf, 1024);
// This listens a responce.
// buf[ret]
= '\0';
printf("Sent Message :
%s",buf);}
close(fd);
exit(0);}
```

As clearly understood by judging from the program, character strings to the line feed are read on a console and the character strings are sent to the descriptor named `fd`. When the file is ready, compile as follows.

```
$ gcc -o talker talker.c
```

Build a server node in HARK from here. What is important is that the timing when a message is sent by a client is unknown so it is necessary to create a server node so that a series of processing is performed every time a message is received Therefore, although the loop calculation for which assumed `Sleep` is assumed as a condition has been performed so far, create a node so that the server itself becomes a trigger of loop processing here. The following is an example.

```
#include <iostream>
#include <BufferedNode.h>
#include <Buffer.h>
#include <string.h>
#include <sstream>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <csignal>
using namespace std;
using namespace FD;
class ListenerTutorial;
DECLARE_NODE(ListenerTutorial);
/*Node
*
* @name ListenerTutorial
* @category HARKD:Tutorial
* @description This block listens to messages from socket and outputs it.
*
* @output_name OUTPUT1
* @output_type string
* @output_description Same as the message received from socket.
*
* @output_name CONDITION
* @output_type bool
* @output_description True if we haven't reach the end of file yet.
*
* @parameter_name PORT
* @parameter_type int
* @parameter_value 8765
* @parameter_description Port number for socket connection
*
END*/
bool exit_flag = false;
class ListenerTutorial :
```

```
public BufferedNode {
int output1ID;
int conditionID;
int port;
int
fd1, fd2;
struct sockaddr_in
saddr;
struct sockaddr_in
caddr;
int
len;
int
ret;
char buf[1024];
public:
ListenerTutorial(string nodeName, ParameterSet params)
:
BufferedNode(nodeName, params)
{output1ID= addOutput("OUTPUT1");
conditionID = addOutput("CONDITION");port= dereference_cast<int>(parameters.get("PORT"));
signal(SIGINT, signal_handler);
signal(SIGHUP, signal_handler);
signal(SIGPIPE, signal_handler);
if ((fd1 = socket(AF_INET, SOCK_STREAM, 0))
< 0)
{perror("socket");exit(1);}
bzero((char *)&saddr, sizeof(saddr));
saddr.sin_family = AF_INET;
saddr.sin_addr.s_addr = INADDR_ANY;
saddr.sin_port = htons(port);
if (bind(fd1, (struct sockaddr *)&saddr, sizeof(saddr))
< 0){perror("bind");exit(1);}
if (listen(fd1, 1)
< 0)
{perror("listen");
exit(1);}
len = sizeof(caddr);
if ((fd2 = accept(fd1, (struct sockaddr *)&caddr, (socklen_t *)&len))
< 0)
{perror("accept");
exit(1);}
close(fd1);
inOrder = true;}
void calculate(int output_id, int count, Buffer &out)
{// Main loop routine starts here.
Buffer &conditionBuffer = *(outputs[conditionID].
buffer);
conditionBuffer[count]= (exit_flag ?
FalseObject :
TrueObject);
ret = read(fd2, buf, 1024);
cout << "Count :
" << count << " , Received Message :
" << buf << endl;
ostringstream message;
message << buf << endl;
string output1 = message.str();
(*(outputs[output1ID].
buffer))
[count]
= ObjectRef(new String(output1));
write(fd2, buf, 1024);
if(exit_flag){
cerr << "Operation closed...
" << endl;
close(fd2);
exit(1);}
// Main loop routine ends here.}
static void signal_handler(int s)
{exit_flag = true;}};
```

Here, the point different from a normal node is that the port that outputs a new bool variable named `CONDITION` is added. This `CONDITION` port always returns `true` except the case of forced termination and pause of socket communication. Use this port as a trigger of a loop of the network file of Flowdesigner. Build a network file. After cutting and pasting the above source, compile and install it. Start Flowdesigner. Create the following node.

Here, see that the `CONDITION` output port becomes `CONDITION` of a loop of the network file. In other words, the loop cycle of this network file is same as the processing cycle of `ListenerTutorial`. The `ListenerTutorial` node

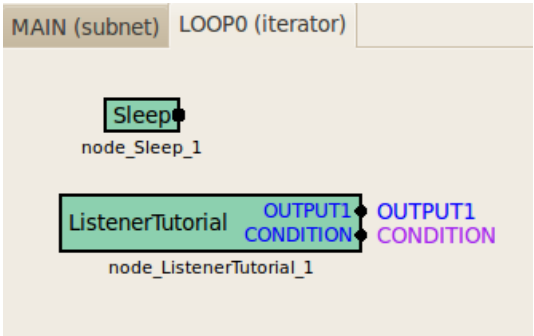


Figure 11.12: Network file: ListenerTutorial

is `ret = read(fd2, buf, 1024);` of the `calculate` function and suspends the processing till messages are received. When the messages are received, all of the `calculate` functions are processed and processing of one `count` is completed. Making `CONDITION` of the network file the `CONDITION` port of the `ListenerTutorial` node enables to perform series of processing accordingly with one receipt of messages. Now, the server node can correspond in accordance with the event of message reception, desired specification is satisfied Now, let's move. First, start the server program (network file of Flowdesigner created above). `Left-click the ListenerTutorial node > Click the "Execute" for the port number in int type` sure to set `CONDITION`. Start the client program next. Start the new console and move to a directory that stores `talker.c` compiled above. Perform the following on a command line. `$ ./talker 127.0.0.1 8765` an IP address is determined `127.0.0.1` assuming that the server and client work with the same machine. Of course it is possible to communicate with a remote machine. Now, enter some characters to the console of `talker`. Then, messages from the client console come to the server node of operating Flowdesigner and they are indicated in each console as follows (e.g. when inputting "`hoge1`", "`hoge2`", "`hoge3`") Server side (`talker.c`)

```
hoge1
Sent Message :
hoge1
hoge2
Sent Message :
hoge2
hoge3
Sent Message :
hoge3
...
```

Client side (Flowdesigner: `ListenerTutorial`))

```
Count :
0 , Received Message :
hoge1
Count :
1 , Received Message :
hoge2
Count :
2 , Received Message :
hoge3
...
```

this way, the node itself of HARK can act as a server and receive a messages from other systems.

11.8 How should I connect with ROS?

Problem

The system is created with ROS, the open source platform for robots developed by Willow Garage Create a robot system that works with speech by passing the obtained sound source localization and the separation results obtained to ROS from HARK. Solution

Since sufficient documents for ROS are available in web, it is assumed for the instruction that (1) the ROS system is already installed and, (2) the basic concepts such as publish to topic or subscribe from topic are know. Concretely, it is assumed that all the Beginner Level of ROS tutorial is already completed. Since there are two connection methods, each of them is described here. Moreover, Python is used for implementation of the ROS node. **(1) Use standard output** Its is a method that executes HARK as a subprocess in a node. utilizing the feature that a network of HARK can be solely executed. Based on the method, localization results from HARK are output as standard outputs (when wishing to have localization results, open the DEBUG property of LocalizeMUSIC). For example, the network `/home/hark/localize.n` can be executed in the python script with the the following code.

```
import subprocess
p = subprocess.Popen( "/home/hark/localize.n"
cwd = "/home/hark/",
stderr = subprocess.STDOUT,
stdout = subprocess.PIPE)
```

Next, outputs of the network can be processed in python scripts with the following code.

```
while not rospy.is_shutdown():
line = p.stdout.readline()
Acquire information from the line
```

Publish to an appropriate topic using the information acquired from HARK. **(2) Use socket communication** Another method is to connect ROS and HARK via socket communication.

It is necessary to create a code of socket communication in the ROS side and a node of that in the HARK side and therefore it is troublesome. However, the entire configuration becomes clear for users. For creation of nodes for HARK, see the tutorial of hark-document. Some part of the python script, which receives information, is shown here.

```
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) sock.bind( ('localhost', 12345)
) sock.listen(1) client, addr = sock.accept()
while 1:
received = client.recv( 1024 )
```

(3) Communicate through topic The last is a communication method utilizing the ROS system. If create a node to do publish of a message for a topic of ROS, localization results and separated sounds can be sent directly to ROS. Discussion

Since (1) is the simplest method, it will be good to try it for the time being. However, when developing a system seriously, (2) or (3) will be better. See Also

1. ROS
2. hark-document tutorial 3 (Creation of node)

11.9 How should I control motor?

Problem

Solution

Since HARK itself is software for audition processing, a module to control a motor is not included. However, when performing comparatively simple control, it is possible to create a module to control a motor. This section describes how to implement the client part of those in HARK as a module by dividing control of a motor into applications of server / client. The following is an example of main processing of the client module.

---

function calculat

1. *if count = 0 then*
2. *setIPaddressandPORTnumber*
3. *connecttoserver*
4. *endif*
5. *obtaininput*
6. *generatecontrolinput*
7. *sendcontrolinput*
8. *checkmotorstate*

---

function destructor

1. *disconnecttoserver*

The first calculate function performs processing related to TCP/IP connection once when it is called for the first time and repeats the same processing in other cases. In order to access to TCP/IP, when this function is called for the first time, it sets the assigned IP address port and connects to the server application (the second and third lines). When it is connected to the server application once, repeats processing to generate a control command from an input is repeated. An input is received first, a command for motor control is created based on the data. The command is sent to the server application and it is confirmed if the command is sent properly. The second destructor function indicates a destructor of a module and performs processing to cut off connection with the server here. The server application receives a control command from the client module and drives the motor based on the command. This server application depends on motors and therefore its description is omitted here.

Discussion

Since Flowdesigner is data flow-oriented, when there is even only node that requires long time for processing, it is difficult to perform a real-time processing in some cases. Therefore, processing on Flowdesigner is kept at generation of a command to control a motor and the load is reduced by performing processing to actually operate a motor in other server applications. Moreover, dividing into a server and a client enables to control another motor only by changing the interface. See Also When actually creating a module for motor control,

"Creation of node" in 12.1 will be useful. Moreover, when combining with control of a more complicated motor or other sensors, "Control of robot" in 12.2 will be helpful.

11.10 How should I copy nodes from other network files?

Problem

I cannot copy and paste a node from other files.

Solution

This problem often occurs when wishing to copy and paste the node that maintains a set point from other files. The reason why copy and paste cannot be performed is because two network files are started in different process.

Copy and paste will be enabled by the following procedure.

1) Start the network file that contains the block to be copied by `"File" -> "Open"`

2) Select the block to be copied by drag,
   `"Edit" -> "copy "`

3) `" Edit " -> "Paste "` at the copy destination

Chapter 12

Others

12.1 What window length and shift length should I use?

Problem

Read this section when wishing to determine an optimal analysis window length and shift length.

Solution

Length is an analysis frame length of speech. Generally, designate a length within 20-40 [ms]. Assuming the sampling frequency as fs Hz, length = fs/1000 * x. x is within 20-40[ms]. Advance is an analysis frame shift length. Generally, designate amount shift amount that overlaps by 1/2 - 1/3 - of the following frame and the entire frame. When performing speech recognition, it is necessary to use the same Length and Advance for acoustic model creation. Discussion

When treating speech, the range in which signals can be assumed to be weakly stationary is 20-40 [ms] and therefore this section describes setting following this policy. A shift length is determined in execution width of a window. Concretely, acquire the window head of a rectangular window with energy equivalent to that of a window function. This window length does not perform frame processing for the same sample redundantly when analyzing continuous frames, and it is possible to perform frame processing without defeat of samples is possible. Since the generally known energy of window functions for speech analyses is energy of about 1/3 - 1/2 in rectangular window length, frame shift amount is used in this range. Although 1/3 is conservative setting and may cause redundant frame processing of the same sample, there are not sample defeats. Although a sample defeat may occur with 1/2 depending on window functions, redundant frame processing does not occur/ However, when using a rectangular window for an analysis, the shift length must be equal to the analysis frame length. In the case of a triangle window, frame shift amount is 1/2.

12.2 Types of windows used for MultiFFT

Problem

Read this section when wishing to know how to choose a window for MultiFFT. Solution

(three kinds: HUMMING, CONJ and RECTANGLE) For speech analysis, choose HUMMING. For other signals, choose appropriately a window used for a spectrum analysis of signals.

12.3 What is MAP?

Problem

Read this when wishing to know about the data types (`Map`$< \cdot\cdot >$) used for inputs and outputs of modules such as MFCCExtractionor SpeechRecognitionClient. Solution

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

12.4 How should I use PreEmphasis?

Problem

I do not know which of the time domain and frequency domain I should use PreEmphasisin. Read this section when you do not know which of the time domain and frequency domain PreEmphasisis to be used for.

Solution

Necessity and effects of PreEmphasis for general speech recognition are described in various books and papers. Please see them PreEmphasiscan be used for both is time domain and frequency domain. However, it is better to choose with considering data used for the acoustic model training.

Chapter 13

Recipes

We show some sample networks.

This chapter introduces the network sample with frequent modules. The sample networks are categorized for each function. The description is summarized simple as possible and therefore it is for training of HARK modules. All standard network configurations are available and therefore the network that the user wishes to create may be already registered as a sample. Therefore, it is recommended to read the category outline of sample networks. Even if there is not the desired network, it is comparatively easier to revise a sample network in the category that is close to the desired network to create the desired network.

13.1 Category of sample network

All categories of sample networks are shown in Table 13.1. There are five categories. The sample networks of each category and files required for execution of the samples are stored in the directories indicated the sample directories on the right column in the table. More than one sample networks are included in the directories. For execution of a sample network, execute a script corresponding to each network file name. Set values of arguments to be given to the network file and necessary setting files are described. For example, if there a network file named demo.n, the script file name that executes the network is demo.sh. Character strings before "." are commonized. However, demo.sh may includes the setting items that depend on operating environments and therefore unexpected result may occur when executing without confirmation. The correct setting method is described in description of each sample network.

Table 13.1: Sample network category.

	Category name	Sample directory name
1	Sound recording network	Record
2	Sound source localization network	Localize
3	Sound source separation network	Separation
4	Acoustic feature extraction network	FeatureExtraction
5	Speech recognition network	Recognition

13.2 Outline of sample network category

Indicate below outline of category of each sample.

- **Sound recording network**
  This is a sample for which modules of the AudioIO category of HARK are used. Monaural sound recording and stereophonic recording are included as basic sound recording samples Stereophonic recording and monaural sound recording operate in most hardware environments. In order to use a sample of multi-channel recording, AD/DA for the multi-channel recording that HARK supports is required. Prepared a sample for the multi-channel recording which radio RASP was used for.

- **Sound source localization network**
  This is a sample for which modules of the Localization category of HARK are used. This is a sample in particular for usage of LocalizeMUSIC. A sample in which a sound source localization result is displayed on a screen with DisplayLocalizationand saved in a file with SaveSourceLocationis available. Recorded sounds for eight channels is available so that sound source localization processing can be confirmed off-line/ Since it is off-line processing, AD/DA is not required and therefore any computers can operate if HRAK is already

installed. In order to execute online localization processing, AD/DA for the multi-channel recording that HARK supports is required. An online sound source localization sample with mike inputs of eight channels by radio RASP is also available.

- **Sound source separation network**
  This is a sample for which modules of the Separation category of HARK are used. This is a sample in particular for usage of GHDSSand PostFilteror GHDSSand HRLE. A sample in which off-line sound source localization processing is performed to recorded sounds for eight channels is available. Since it is off-line processing, AD/DA is not required and therefore any computers can operate if HRAk is already installed. In order to execute online localization processing, AD/DA for the multi-channel recording that HARK supports is required. An online sound source separation sample with mike inputs of eight channels by radio RASP is also available.

- **Acoustic feature extract network**
  This is a sample in which modules of the FeatureExtraction category of HARK are used. This is a sample in particular for usage of MSLSExtractionand MFCCExtraction. A sample in which off-line acoustic feature extract is performed to recorded sounds for one channel is available.

- **Speech recognition network**
  This is a sample in which ASR of HARK and modules of the MFT category are used. This is a sample in particular for usage of MFMGenerationand SpeechRecognitionClient. A sample in which off-line sound source localization processing is performed to recorded sounds for eight channels is available. When the radio RASP is connected, a sample with mike inputs of eight channels by radio RASP is also available.

13.3 Notation of document and method for execution of sample network

An operation example of the description is shown in the rectangular region. > on the line head indicates a command prompt. The bold-faced part of the operation example indicates an input from the user, and the italic face part indicates a message from the system. In the following example, ">" in the first line indicates a command prompt. Note that the prompt is displayed in different ways (e.g. "%", "$") according to operating

Figure 13.1: Execution example of sample network start-up script

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

environments. The bold letters that come after the prompt in the first line are the letters entered by the user. In the above example, the seventeen letters of echo "HelloWorld!" are the letters entered by the user. Press the enter key at the end of line. The italic letters in the second line are the output from the system, which are displayed as pressing the enter key at the end of the first line.

13.4 Sound recording network sample

Three kinds of sound recording network samples are available as shown in Table 13.2. The left column indicates network files, the middle column indicates shell script files to operate the networks and the right column indicates processing contents.

Table 13.2: Sound recording network list.

Network file name	Network execution script	Processing content
demoALSA1ch.n	demoALSA1ch.sh	Monaural sound recording
demoALSA2ch.n	demoALSA2ch.sh	Stereo sound recording
demoWS8ch.n	demoWS8ch.sh	Sound recording for eight channels with RASP

13.4.1 Monaural sound recording

A sample of monaural sound recording is shown here. Deepen understanding while executing the sample. Execute demoALSA1ch.sh in the Record directory. Designate the number of frames for the speech to be collected as an argument. In this example, 0.01 frames per sec. Designate 100 here and record for one second. When sound recording is completed, a file named rec0.sw.wav is generated. Flow of generation of the rec0.sw.wav file by the

script is shown. In demoALSA1ch.sh, the first argument that the script receives is passed to demoALSA1ch.n. demoALSA1ch.n records for the number of frames designated at a sampling rate of 16,000Hz and generate a file named rec0.sw in the 16-bit little endian format with a code. The script then gives a header in the RIFF format to rec0.sw using a sox command. It finally generates a file named rec0.sw.wav and erases rec0.sw at the end. An execution example of demoALSA1ch.sh is shown in Figure 13.2. After the execution, the sound is recorded for about one second and the file ref0.sw.wav is generated. When replaying this file, the recorded content can be confirmed. Recording cannot be performed well, perform the following checks.

Figure 13.2: Execution example of demoALSA1ch.sh.

```
> demoALSA1ch.sh 100
UINodeRepository:: Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN

<Bool 1 >
```

1. The microphones are properly connected. Confirm if the plugs are not unplugged or loosened and connect them properly.
2. Check if the microphone terminals of the computer accept plug in power. If they do not accept plug in power, it is necessary to supply a power to the microphones. For battery type microphones, set batteries and switch on. For battery box type microphones, connect battery boxes and switch on.
3. Confirm if the sound is replayed or recorded with software other than HARK. If the sound cannot be replayed or recorded, the OS and driver, and audio interface may not be properly set or connected so check them.
4. When more than two audio interfaces are connected, remove audio interfaces after the second ones before recording the sound. If recording cannot be performed with one audio interface, change the property of demo.n. For the DEVICE property of the AudioStreamFromMicmodule, try setting of 0,0 plughw: 0,1 plughw: 0,2.
5. Comment out the rm command in the last line of the script and confirm that rec0.sw is generated. When rec0.sw.wav is not generated though rec0.sw is generated, install SOX.

Five modules are included in this sample. There is one module in MAIN (subnet) and are four modules in MAINLOOP (iterator). MAIN (subnet) and MAINLOOP (iterator) are shown in Figure 13.3 and 13.4, respectively. It is simple network configuration in which the audio waveforms collected in the AudioStreamFromMicmodule are written in a file in SaveRawPCM. Data formats between modules are uniformed through MatrixToMap. Iterate is a module that indicates that the execution is repeated for the number of times set by the user. Designate the number of frames to be collected and adjust sound recording time. There are five properties for the MAINLOOP

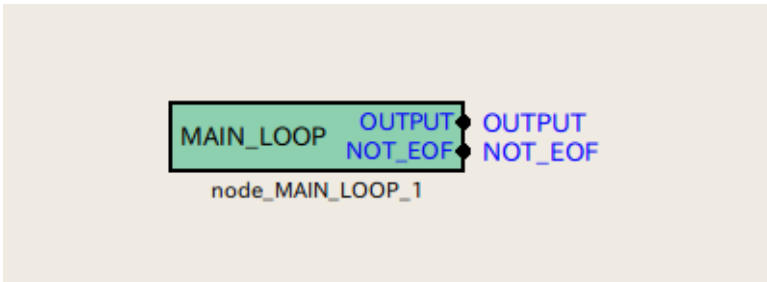


Figure 13.3: MAIN (subnet)

module in MAIN (subnet). Table 13.3 shows a list of them. SAMPLINGRATE and GETFRAMES are important. Each parameter value is set according to the values in the table. Set value of GETFRAMES is `int`:ARG1 here. This means the monobasic argument of demoALSA1ch.n casted to the integral type and substituted. The number of sound recording frames is given to this argument in demoALSA1ch.n in demoALSA1ch.sh. Designate

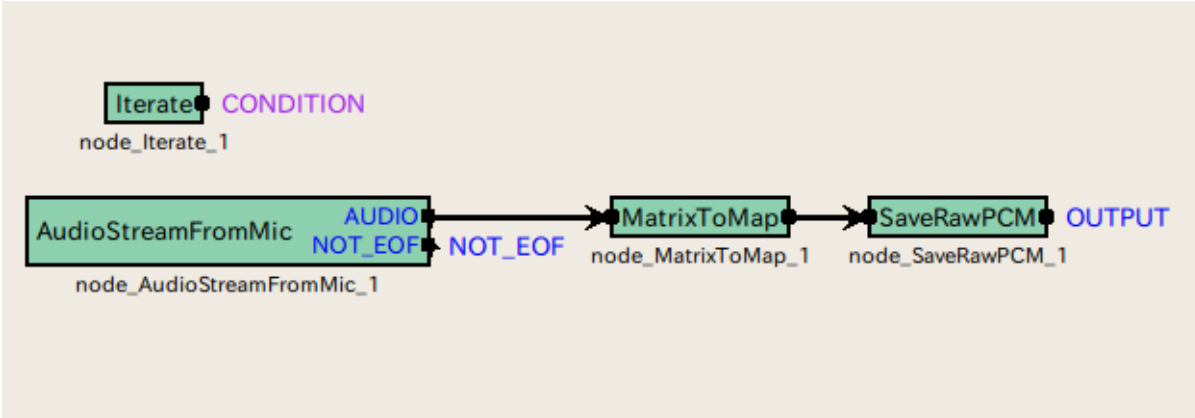


Figure 13.4: MAINLOOP (iterator)

sound recording time length as the number of frames acquired and the actual sound recording time length is expressed in sec as follows.

$$(LENGTH + (GETFRAMES - 1) * ADVANCE)/SAMPLINGRATE \tag{13.1}$$

Among the four modules in MAINLOOP (iterator), an important module is AudioStreamFromMic. Parameters

Table 13.3: Parameter list of **MAINLOOP**

Parameter name	Type	Set value	Unit	Description
ADVANCE	int	160	[pt]	Shift Length
LENGTH	int	512	[pt]	FFT length
SAMPLING_RATE	int	16000	[Hz]	Sampling frequency
GETFRAMES	int	int:ARG1		Number of framse for sound recording
DOWHILE	bool			Blank

are shown in Table 13.4. In this module, designate a sound recording device. Designate a sound recording device in this module. In this demo, the number of sound recording channels is 1ch so that a lot of users can check the operation, and ALSA is designated for DEVICETYPE and plughw: 0,0 is designated for DEVICE.

Table 13.4: Parameter list of AudioStreamFromMic

Parameter name	type	Set value	Unit	Description
LENGTH	subnetparam	LENGTH	[pt]	FFT length
ADVANCE	subnetparam	ADVANCE	[pt]	Shift length
CHANNEL_COUNT	int	1	[ch]	Number of sound recording channel
SAMPLING_RATE	subnetparam	16000	[Hz]	Sampling frequency
DEVICETYPE	string	ALSA		Device type
DEVICE	string	plughw:0,0		Device name

13.4.2 Stereo recording

A sample of stereo recording is shown here. Deepen understanding while executing the sample. Execute demoALSA2ch.sh in the Record directory. Designate the number of frames for the speech to be collected as an argument. In this example, 0.01 frames per sec. Designate 100 here and record for one second. When the sound recording is completed, files named rec0.sw.wav and rec1.sw.wav are generated. Flow of generation of the rec0.sw.wav file by the script is shown. In demoALSA2ch.sh, the first argument that the script receives is passed to demoALSA2ch.n. demoALSA2ch.n records for the number of frames designated at a sampling rate of 16,000Hz and generate files named rec0.sw and rec1.sw in the 16-bit little endian format with a code. The script then gives a header in the RIFF format to rec0.sw and rec1.sw using a sox command. It finally generates files named rec0.sw.wav and rec1.sw.wav and erases rec0.sw and rec1.sw at the end. An execution example of demoALSA2ch.sh is shown in Figure 13.5. After the execution, the sound is recorded for about one second and the files rec0.sw.wav and rec1.sw.wav are generated. When replaying this file, the recorded content can be confirmed. Some computer rarely do not accept stereo recording so an error may occur. In the case that the

Figure 13.5: Execution example of demoALSA2ch.sh

```
> demoALSA2ch.sh 100
UINodeRepository:: Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN

<Bool 1 >
```

sound recording cannot be performed well even though the computer is able to accept stereo recording, confirm the check items for monaural sound recording. Five modules are included in this sample. There is one module in MAIN (subnet) and are four modules in MAINLOOP (iterator). MAIN (subnet) and MAINLOOP (iterator) are shown in Figures 13.6 and 13.7. It is simple network configuration in which the audio waveforms collected in the AudioStreamFromMicmodule are written in a file in SaveRawPCM. Data formats between modules are uniformed through MatrixToMap. Iterate is a module that indicates that the execution is repeated for the number of times set by the user. Designate the number of frames to be collected and adjust sound recording time. What is different from the monaural sound recording network is the only CHANNELCOUNT property of the AudioStreamFromMicmodule. It is 1 for monaural recording and 2 for stereo recording.

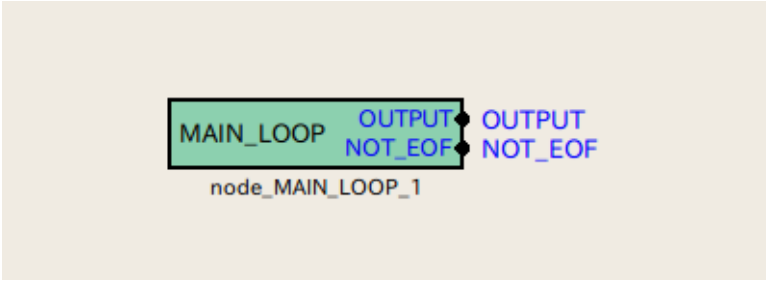


Figure 13.6: MAIN (subnet)

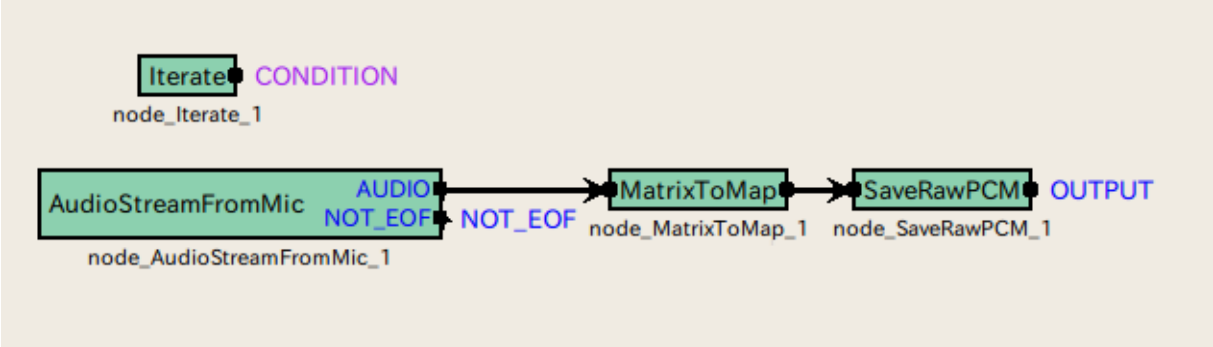


Figure 13.7: MAINLOOP (iterator)

13.4.3 8ch sound recording with radio RASP

8ch sound recording is introduced as a sample of multi-channel sound recording. Deepen understanding while executing the sample. Execute demoWS8ch.sh in a Record directory, after setting several items. demoWS8ch.sh executes demoWS8.n inside. Although 127.0.0.1 is designated as an IP address of the radio RASP in demoWS8ch.n, it is necessary to change this to an appropriate IP address. If configuration of FPAA for the radio RASP is not completed yet, complete it here. See the document of radio RAS for the method to execute wsfpaaconfig. When

it is ready, Execute demoWS8ch.sh included in the Record directory. Designate the number of frames for the speech to be collected in demoWS8ch.n as an argument in demoWS8ch.sh. In this example, 0.01 frames per sec. Designate 100 here and record for one second. Open demoWS8ch.n in flowdesigner and open the LOOP subnetwork tab. Then, open the DEVICE property of the AudioStreamFromMicmodule and change. When sound recording is completed, files named rec0.sw.wav, rec1.sw.wav, ..., rec7.sw.wav are generated. Flow of generation of the rec0.sw.wavrec1.sw.wav... rec7.sw.wav files by the script is shown. In demoWS8ch.sh, the monobasic argument that the script receives is passed to demoWS8ch.n. This argument indicates the number of frames to be recorded. demoWS8ch.n records for the number of frames designated at a sampling rate of 16,000Hz and generate files named rec0.sw, rec1.sw, ..., rec7.sw in the 16-bit little endian format with a code. The script then gives a header in the RIFF format to rec0.sw, rec1.sw, ..., rec7.sw using a sox command. It finally generates files named rec0.sw.wav, rec1.sw.wav, ..., rec7.sw.wav and erases rec0.sw, rec1.sw, ..., rec7.sw at the end. An execution example of demoALSA2ch.sh is shown in Figure 13.8. After the execution, the sound is recorded for about one second and the files rec0.sw.wav, rec1.sw.wav, ..., rec7.sw.wav are generated. When replaying this file, the recorded content can be confirmed. When recording cannot be performed well, perform the following checks.

Figure 13.8: Execution example of demoWS8ch.sh.

```
> demoWS8ch.sh 100
UINodeRepository::Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN

<Bool 1 >
```

1. Check if the microphones are properly connected. Check if the plugs are not unplugged or loosened and connect them properly.
2. The network connection is established with the IP address of RASP. Check the network connection with ping.
3. A correct IP address of RASP is set to AudioStreamFromMic.
4. Initialization of FPAA of RASP is completed.
5. Confirm if the sound is replayed or recorded with software other than HARK. If the sound cannot be replayed or recorded, the OS and driver, and audio interface may not be properly set or connected so check them.
6. Comment out the rm command in the last line of the script and confirm that rec0.sw is generated. When rec0.sw.wav is not generated though rec0.sw is generated, install SOX.

Six modules are included in this sample. There is one module in MAIN (subnet) and are five modules in MAINLOOP (iterator). MAIN (subnet) and MAINLOOP (iterator) are shown in Figures 13.9 and 13.10. It is simple network configuration in which the audio waveforms collected in the AudioStreamFromMicmodule are written in a file in SaveRawPCM. The number of channels are selected in in ChannelSelector. It is necessary to use the AudioStreamFromMicmodule t the 16ch sound recording mode when connected to a radio RASP. Therefore, in order to record sounds for eight channels, select out eight channels within the sixteen channels. In this example, 0-7ch are selected in ChannelSelector. Data formats between modules are uniformed through MatrixToMap. Iterate is a module that indicates that the execution is repeated for the number of times set by the user. Designate the number of frames to be collected and adjust sound recording time. There are five properties for the MAINLOOP module in MAIN (subnet). Parameters are shown in Table 13.5. SAMPLINGRATE and GETFRAMES are important. Each parameter value is set according to this table. Set value of GETFRAMES is `int`:ARG1 here. This means the monobasic argument of demoALSA1ch.n casted to the integral type and substituted. The number of sound recording frames is given to this argument in demoALSA1ch.n in demoALSA1ch.sh. Designate sound recording time length as the number of frames acquired and the actual sound recording time length is expressed in sec as follows.

$$(LENGTH + (GETFRAMES - 1) * ADVANCE)/SAMPLINGRATE \tag{13.2}$$

Among the five modules in MAINLOOP (iterator), an important module is AudioStreamFromMic. Parameters are shown in Table 13.6. Designate a sound recording device in this module. Designate a multi-channel sound recording device. Set 16ch to the number of sound recording channel, designate WS, which indicates radio RASP, in DEVICETYPE and an IP address in DEVICE. In order to execute the sample, it is necessary to change the address to that of the actual radio RASP.

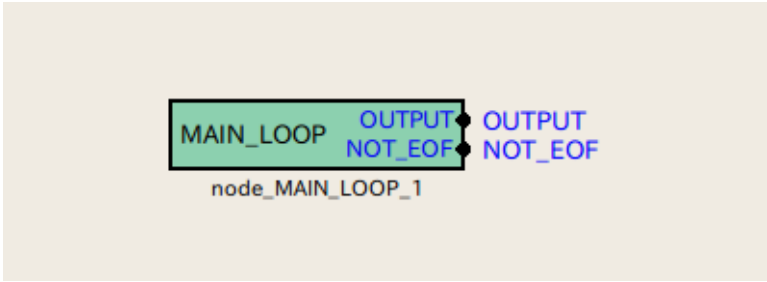


Figure 13.9: MAIN (subnet)

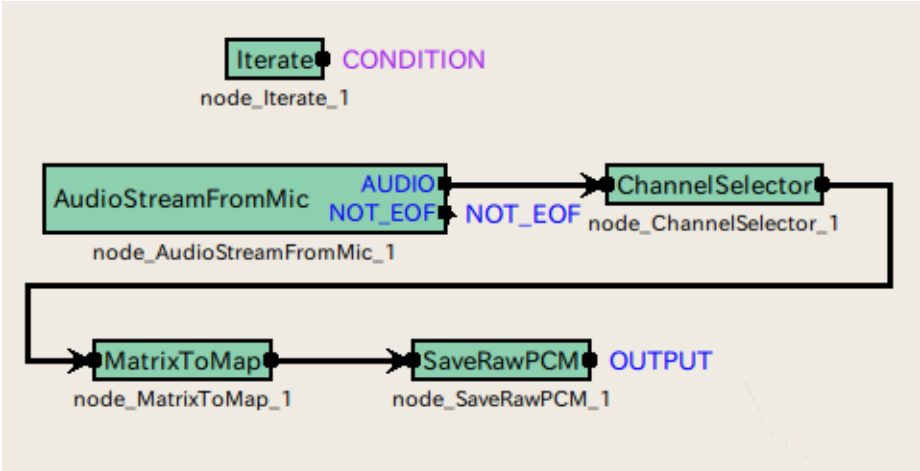


Figure 13.10: MAINLOOP (iterator)

Table 13.5: Parameterlist of **MAINLOOP**

Parameter name	Type	Set value	Unit	Description
ADVANCE	int	160	[pt]	Shift length
LENGTH	int	512	[pt]	FFT length
SAMPLINGRATE	int	16000	[Hz]	Sampling frequency
GETFRAMES	int	int:ARG1		Number of frames for sound recording
DOWHILE	bool			Blank

Table 13.6: Parameter list of AudioStreamFromMic

Parameter Name	Type	set value	Unit	Description
LENGTH	subnetparam	LENGTH	[pt]	FFT length
ADVANCE	subnetparam	ADVANCE	[pt]	Shift length
CHANNELCOUNT	int	16	[ch]	Number of channels for sound recording
SAMPLINGRATE	subnetparam	16000	[Hz]	Sampling frequency
DEVICETYPE	string	WS		Device type
DEVICE	string	127.0.0.1		Device name

13.5 Network sample of sound source localization

There are two kinds of network samples of sound source localization as shown in Table 13.7. The left column indicates network files, the middle column indicates shell script files to operate the networks and the right column indicates processing contents.

Table 13.7: Sound source localization network list.

Network file name	Network execution script	Processing content
demoOffline8ch.n	demoOffline8ch.sh	Sound source localization processing of 8ch audio file
demoWS8ch.n	demoWS8ch.sh	8ch on-line source localization processing with radio RASP

13.5.1 Off line sound source localization

A sample of off line sound source localization is introduced first. Since input speeches are files, even the users who do not have a multi-channel AD/DA can confirm while executing sound source localization processing. Execute demoOffline8ch.sh in the Localization directory. An execution example is shown in Figure 13.11. After execution,

Figure 13.11: Execution example of demoOffline8ch.sh.

```
> demoOffline8ch.sh
Display localization result of sound source in the 000 degrees direction
UINodeRepository::Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN
reading A matrix
Identifier is different from HARK's one:H
Try to read ../config/music.dat as header-less MUSIC transfer function format.
72 directions, 1 ranges, 8 microphones, 512 points
done
initialize
Source 0 is created.
Source 0 is removed.
Omitted
Source 9 is created.
Source 9 is removed.
Vector<ObjectRef>
Vector<ObjectRef>
Display localization result of sound source in the 090 degrees direction
The rest is omitted
```

a sound source localization result is indicated. At the same time, a sound source localization result is output to a file in the text format. In the script, four 8ch audio files are read in a sequential order and its localization results are displayed. The files are of sound sources in the 0,90,180 and 270 degrees direction. The sound source localization results are saved in the text files of Localization000.txt, Localization090.txt, Localization180.txt and Localization270.txt for directions of each sound source. For description of sound source localization result files, see "Format" section in the hark-document. In this sample, the input file to the sound source localization network is a virtually synthesized sound in a known direction. When localization cannot be performed well, check the following items.

1. Check if f101000.wav, f101090.wav, f101180.wav and f101270.wav are in the ../data directory. These files are simulation synthetic sounds of the 8ch sound recording audio waveforms to which impulse responses from the 0, 90, 180 and 270 degrees direction are superimposed. They are interpreted as sounds obtained in the case sounds are given to a virtual robot from each direction. Sound source localization results are not displayed without these files.

2. Check if the music.dat files are in the ../config directory. They are the impulse response files of a virtual robot. Sound source localization results are not displayed without these files.

3. Check if gnuplot is installed. Sound source localization results are displayed in gnuplot. The processing is ended if gnuplot cannot be executed. Check if gnuplot can be executed and set a path to gnuplot if it cannot be executed. If gnuplot is not installed, install it and set a path.

Ten modules are included in this ample. There is three modules in MAIN (subnet) and are seven modules in MAINLOOP (iterator). MAIN (subnet) and MAINLOOP (iterator) are shown in Figures **??** and **??**. The audio waveforms which collected with AudioStreamFromWavemodule are analyzed in MultiFFTand sound source localization is performed in LocalizeMUSIC. It is a netwok structure in which Localization results are tracked with SourceTrackerand SourceIntervalExtender, displayed in DisplayLocalizationand written in files in SaveSourceLocation. There are six parameters for the properties of the MAINLOOP module in MAIN (subnet). Its parameter list is

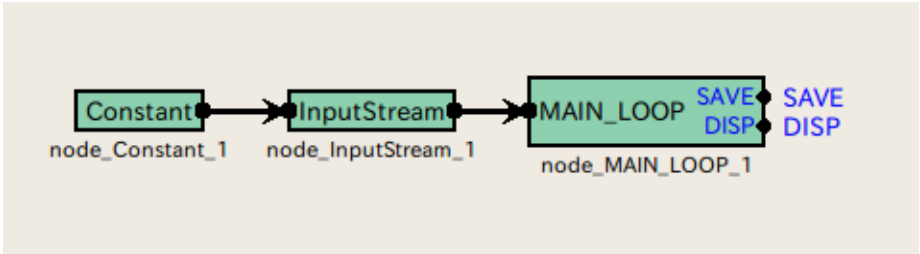


Figure 13.12: MAIN (subnet)

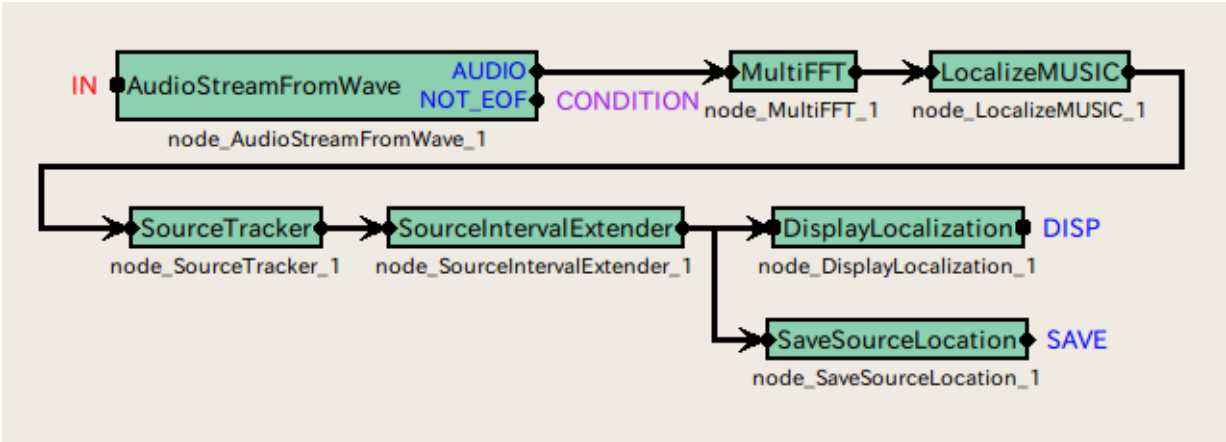


Figure 13.13: MAINLOOP (iterator)

shown in Table 13.8. Each parameter value is set according to this table. An important parameter is AMATRIX. Files created in harktool3 from impulse responses of a robot are used. Sound source localization cannot be performed with files for other robots. Among the seven modules in MAINLOOP (iterator), an important module

Table 13.8: Parameter list of **MAINLOOP**

Parameter name	Type	Set value	Unit	Description
LENGTH	int	512	[pt]	FFT length
ADVANCE	int	160	[pt]	Shift length
SAMPLINGRATE	int	16000	[Hz]	Sampling frequency
AMATRIX	int	ARG2		AMATRIX file name
FILENAME	subnetparam	ARG3		Name of file in which sound source localization result is written
SPEEDOFSOUND	float	343.0	[m/s]	Acoustic velocity
DOWHILE	bool			Blank

is LocalizeMUSIC. Parameters are shown in Table 13.9.

13.5.2 Online sound source localization

A sample of online sound source localization is introduced here. Since this sample cannot be executed without a multi-channel AD/DA, execute after preparing AD/DA. For execution of the sample, set an IP address of the

Table 13.9: Parameter list of LocalizeMUSIC

Parameter name	Type	Set value	Unit	Description
NUMCHANNELS	int	8	[ch]	Number of channels
LENGTH	subnetparam	LENGTH	[pt]	FFT length
SAMPLINGRATE	subnetparam	SAMPRINGRATE	[Hz]	Sampling frequency
AMATRIX	subnetparam	AMATRIX		AMATRIX file name
PERIOD	int	50		Smoothing time width of correlation matrix
NUMSOURCE	int	1		Number of sound sources
MINDEG	int	-180	[deg]	Minimum localization angle
MAXDEG	int	180	[deg]	Maximum localization angle
LOWERBOUNDFREQUENCY	int	500	[Hz]	Lower limit frequency of processing object
HIGHERBOUNDFREQUENCY	int	2800	[Hz]	Upper limit frequency of processing object
DEBUG	bool	`false`		Display / non-display of debug information

radio RASP to demoWS8ch.n and execute demoWS8ch.sh in the Localization directory. 127.0.0.1 is set to it beforehand. Change this to the actual IP address of the radio RASP. If configuration of FPAA of the radio RASP is not completed yet, execute it here. For the execution method, see the document of the radio RASP. When the above is completed, execute demoWS8ch.sh in the Localization directory. An execution example is shown in Figure 13.14. After the execution, a sound source localization result is indicated. When localization

Figure 13.14: Execution example of demoWS8ch.sh.

```
> demoWS8ch.sh
UINodeRepository::Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN
reading A matrix
Identifier is different from HARK's one:H
Try to read ../config/music.dat as header-less MUSIC transfer function format.
72 directions1 ranges8 microphones512 points
done
initialize
Source 0 is created.
Source 0 is removed.
The rest is omitted
```

cannot be performed well, check the following items.

1. Check if the music.dat files are in the ../config directory. They are impulse response files of a virtual robot. Sound source localization results are not indicated without these files.

2. Check if gnuplot is installed. Sound source localization results are displayed in gnuplot. The processing is ended if gnuplot cannot be executed. Check if gnuplot can be executed and set a path to gnuplot if it cannot be executed. If gnuplot is not installed, install it and set a path.

3. MAINLOOP (iterator) Adjust values of the THRESH property of SourceTrackerin MAINLOOP (iterator). When no sound source localization results are displayed, lower the values. When sound source localization results are displayed though sounds other than the sound sources are localized, raise the values. When it is difficult to adjust the values, set the DEBUG property of LocalizeMUSICto `true`and input sounds from the front of the microphone array. MUSIC spectra are indicated in the terminal. Confirm values of the sound source directions and those of other directions and adjust THRESH in the range.

4. Check if the microphones are connected properly. Check if the plugs are not unplugged or loosened and connect them properly.

5. Check if the microphone terminals of the computer accept plug in power. If they do not accept plug in power, it is necessary to supply a power to the microphones. For battery type microphones, set batteries and switch on. For battery box type microphones, connect battery boxes and switch on.

6. Confirm if the sound is replayed or recorded with software other than HARK. If the sound cannot be replayed or recorded, the OS and driver, and audio interface may not be properly set or connected so check them.

7. When more than two audio interfaces are connected, remove audio interfaces after the second ones before recording the sound. If recording cannot be performed with one audio interface, change the property of demo.n. For the DEVICE property of the AudioStreamFromMicmodule, try setting of 0,0 plughw: 0,1 plughw: 0,2.

8. Comment out the rm command in the last line of the script and confirm that rec0.sw is generated. When rec0.sw.wav is not generated though rec0.sw is generated, install SOX.

Eight modules are included in this sample. There is one module in MAIN (subnet) and are seven modules in MAINLOOP (iterator). MAIN (subnet) and MAINLOOP (iterator) are shown in Figures **??** and 13.16. Channels used for the audio waveforms collected with the AudioStreamFromMicmodule in ChannelSelectorare selected and they are converted into spectral representation in MultiFFT, and MUSIC spectra are obtained in LocalizeMUSIC. Next, peaks of the MUSIC spectra are obtained in SourceTrackerand tracking is performed based on the continuity with a peak of the former processing time. SourceIntervalExtenderis processing to extend the start time of the sound source to the past. Finally, sound source localization results are displayed in DisplayLocalization. The

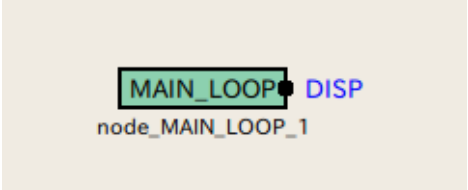


Figure 13.15: MAIN (subnet)

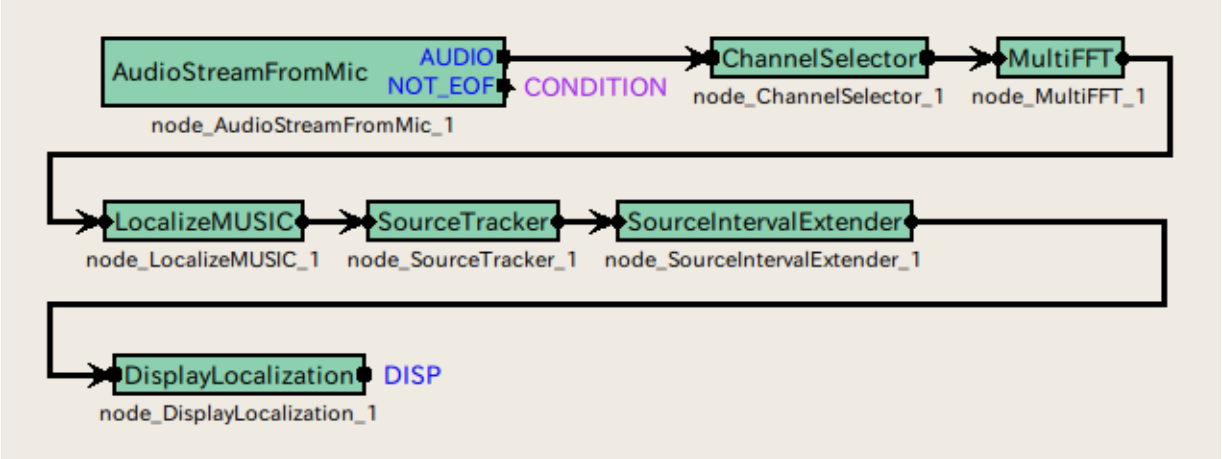


Figure 13.16: MAINLOOP (iterator)

property of the MAINLOOP module in MAIN (subnet) has five parameters. Its parameter list is shown in Table 13.10. Each parameter value is set according to this table. An important parameter is AMATRIX. Use the file which it created in harktool3 from impulse response of a robot. Among the seven modules in MAINLOOP

Table 13.10: Parameter list of **MAINLOOP**

Parameter name	Type	Set value	Unit	Description
LENGTH	`int`	512	[pt]	FFT length
ADVANCE	`int`	160	[pt]	Shift Length
SAMPLINGRATE	`int`	16000	[Hz]	Sampling frequency
AMATRIX	`int`	ARG1		AMATRIX file name
DOWHILE	`bool`			Blank

(iterator), an important module is LocalizeMUSIC. Parameters are shown in Table 13.11.

Table 13.11: Parameter list of LocalizeMUSIC

Parameter name	Type	Set value	Unit	Description
NUMCHANNELS	int	8	[ch]	Number of channels
LENGTH	subnetparam	LENGTH	[pt]	FFT length
SAMPLINGRATE	subnetparam	SAMPRINGRATE	[Hz]	Sampling frequency
AMATRIX	subnetparam	AMATRIX		AMATRIX file name
PERIOD	int	50		Smoothing time width of correlation matrix
NUMSOURCE	int	1		Number of sound sources
MINDEG	int	-180	[deg]	Minimum localization angle
MAXDEG	int	180	[deg]	Maximum localization angle
LOWERBOUNDFREQUENCY	int	500	[Hz]	Lower limit frequency of processing object
HIGHERBOUNDFREQUENCY	int	2800	[Hz]	Upper limit frequency of processing object
DEBUG	bool	false		Display / non-display of debug information

13.6 Network sample of sound source separation

There are three kinds of network samples for sound source separation as shown in Table 13.12. The left column indicates network files, the middle column indicates shell script files to operate the networks and the right column indicates processing contents.

Table 13.12: Sound source separation network list.

Network file name	network execution script	Processing content
demoOffline8ch.n	demoOffline8ch.sh	Sound source separation processing of 8ch audio file
demoWS8ch.n	demoWS8ch.n	8ch on-line sound source separation processing with radio RASP
demoOfflineHRLE.n	demoOfflineHRLE.sh	Processing for which noise estimation by HRLE as a postprocessing and su

13.6.1 Off line sound source separation

A sample of off line sound source separation is introduced first. Since input sounds are files, even the users who do not have a multi-channel AD/DA can confirm while executing sound source localization processing. Execute demoOffline8ch.sh in the Localization directory. An execution example is shown in Figure 13.17. After

Figure 13.17: Execution example of demoOffline8ch.sh.

```
> demoOffline8ch.sh
UINodeRepository:: Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN
TF = 1,INITW = 0,FixedNoise = 0
SSMethod = 0LCCONST = 0LCMETHOD = 0
NF =257NSrc = 72NMic = 8 Data =
Location =
Robot =
ArrayType =
Creator =
Memo =
```

the execution, sound sources are separated and separated sounds are generated. When separation cannot be performed well, check the following items.

1. Check if the ghdss.dat files are in the ../config directory. They are impulse response files of a virtual robot. Sound source localization results are not indicated without these files.

   Check if the sample2spdat files are in the ../data directory. These files are simulation synthetic sounds of the 8ch sound recording audio waveforms to which impulse responses from the -60 and 60 degrees direction are superimposed. They are interpreted as sounds obtained in the case sounds are given to a virtual robot

from each of the -60 and 60 degrees direction simultaneously. If these files are not present, it means there are no inputs for sound source separation and therefore sound source separation cannot be performed.

Nine modules are included in this sample. There are three modules in MAIN (subnet) and are six modules in MAINLOOP (iterator). MAIN (subnet) and MAINLOOP (iterator) are shown in Figures 13.18 and 13.19. The audio waveforms read from the files in the AudioStreamFromWavemodule are analyzed in MultiFFT, separated in GHDSS, synthesized in Synthesizeand the audio waveforms are saved in SaveRawPCM. ConstantLocalization, which outputs the information that sound sources are localized from the designated direction constantly, is used for sound source localization. When replace this module with LocalizeMUSICin accordance with the sample of Localization, sound source localization is performed at the same time. The property of the MAINLOOP module

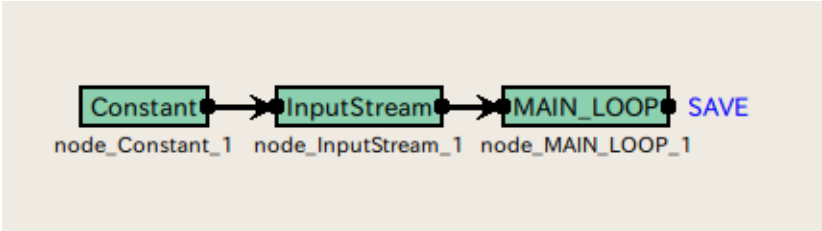


Figure 13.18: MAIN (subnet)

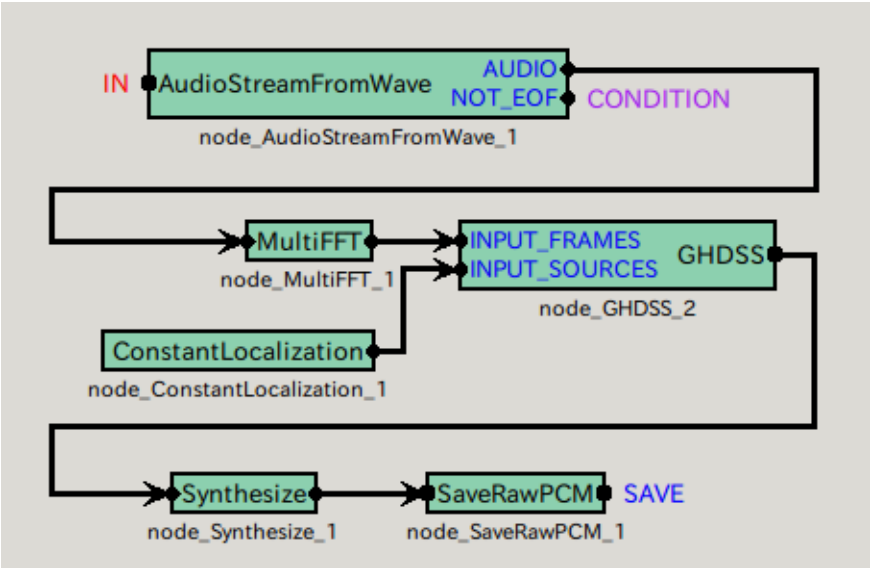


Figure 13.19: MAINLOOP (iterator)

in MAIN (subnet) has five parameters. Its parameter list is shown in Table 13.13. Each parameter values is set according to this table. An important parameter is TFCONJFILENAME. Use the file created in harktool3 from impulse responses of a robot. Among the six modules in MAINLOOP (iterator), an important module is GHDSS.

Table 13.13: Parameter list of **MAINLOOP**

Parameter name	Type	Set value	Unit	Description
LENGTH	int	512	[pt]	FFT length
ADVANCE	int	160	[pt]	Shift length
SAMPLINGRATE	int	16000	[Hz]	Sampling frequency
TFCONJFILENAME	int	ARG2		TFCONJ file name
DOWHILE	bool			Blank

Its parameters are shown in Table 13.14.

Table 13.14: Parameter list of GHDSS

Parameter name	Type	Set value	Unit	Description
LENGTH	subnetparam	LENGTH	[pt]	FFT length
ADVANCE	subnetparam	ADVANCE	[pt]	Shift length
SAMPLINGRATE	subnetparam	SAMPRINGRATE	[Hz]	Sampling frequency
LOWERBOUNDFREQUENCY	int	0	[Hz]	Lower limit frequency of processing object
HIGHERBOUNDFREQUENCY	int	8000	[Hz]	Upper limit frequency of processing object
TFCONJ	string	DATABASE		Mode that uses measured value
TFCONJFILENAME	subnetparam	TFCONJFILENAME		TFCONJ file name
FIXEDNOISE	bool	false		No directivity-fixed noise source
INITWFILENAME	string			Initial value of W is not given
SSMETHOD	string	ADAPTIVE		Adaptation calculation
SSSCAL	float	1.0		
NOISEFLOOR	float	0		
LCCONST	string	FULL		Update all elements of separation matrix
LCMETHOD	string	ADAPTIVE		Adaptation calculation
UPDATEMETHODTFCONJ	string	POS		Update TFCONJ for each direction
UPDATEMETHODW	string	ID		Update W for each sound source ID
UPDATEACCEPTANGLE	float	0.1		Permissive angle error
EXPORTW	bool	false		Do not output W
UPDATE	string	STEP		Update for each update step

13.6.2 Online sound source separation

A sample of online sound source separation is introduced here. Since this sample cannot be executed without a multi-channel AD/DA, execute after preparing AD/DA. For execution of the sample, set an IP address of the radio RASP to demoWS8ch.n and execute demoWS8ch.sh in the Sample directory. 127.0.0.1 is set to it beforehand. Change this to the actual IP address of the radio RASP. If configuration of FPAA of the radio RASP is not completed yet, execute it here. For the execution method, see the document of the radio RASP. When the above is completed, execute demoWS8ch.sh in the Localization directory. An execution example is shown in Figure 13.20. After the execution, the sounds are separated and separated sounds are generated. Separated sounds

Figure 13.20: Execution example of demoWS8ch.sh.

```
> demoWS8ch.sh
UINodeRepository::Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN
TF = 1,INITW = 0,FixedNoise = 0
SSMethod = 0LCCONST = 0LCMETHOD = 0
NF =257NSrc = 72NMic = 8 Data =
Location =
Robot =
ArrayType =
Creator =
Memo =
```

continue to be generated till pressing the Control key and 'c' key at the same time After appropriate time passed, terminate the separation by pressing the Control key and 'c' key at the same time. When separation cannot be performed well, check the following items.

1. Check if the ghdss.dat files are in the ../config directory. They are impulse response files of a virtual robot. Sound source localization results are not indicated without these files.

2. Check if the microphones are connected properly. Check if the plugs are not unplugged or loosened and connect them properly.

3. BCheck if network connection is established with the IP address of RASP. Check the network connection with ping.

4. Check if the correct RASP IP address is set for AudioStreamFromMic.
5. Initialization of FPAA of RASP is completed.
6. Confirm if the sound is replayed or recorded with software other than HARK. If the sound cannot be replayed or recorded, the OS and driver, and audio interface may not be properly set or connected so check them.
7. Comment out the rm command in the last line of the script and confirm that rec0.sw is generated. When rec0.sw.wav is not generated though rec0.sw is generated, install SOX.

??) modules are included in this sample. There is one module in MAIN (subnet) and are ?? modules in MAINLOOP (iterator) MAIN (subnet) and MAINLOOP (iterator) are shown in 13.21 and 13.22. The audio waveforms collected in the AudioStreamFromMicmodule are analyzed in MultiFFT, the sounds are separated in GHDSSand are synthesized in Synthesize, and the audio waveforms are saved in SaveRawPCM. ConstantLocalization, which outputs the information that sound sources are localized from the designated direction constantly, is used for sound source localization. Changing this module to LocalizeMUSICin accordance with the sample of Localization and enables to perform sound source localization at the same time. The property of the MAINLOOP module

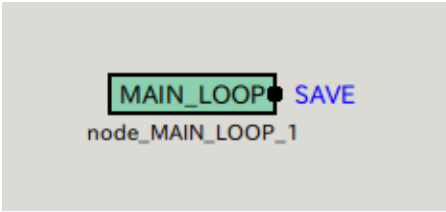


Figure 13.21: MAIN (subnet)

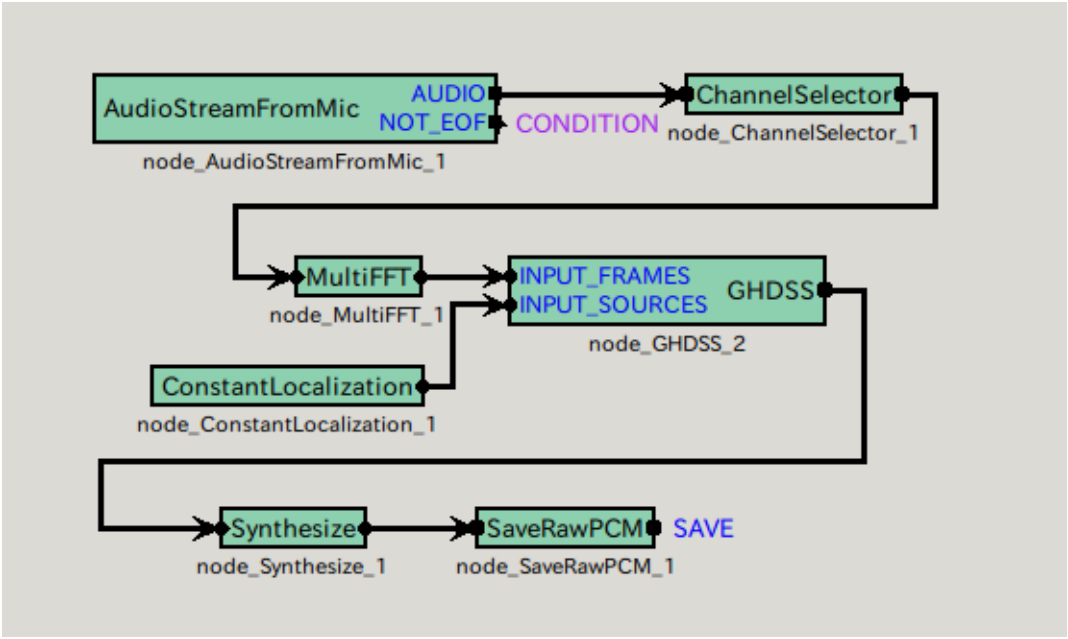


Figure 13.22: MAINLOOP (iterator)

in MAIN (subnet) has five parameters. Its parameter list is shown in Table 13.15. Each parameter value is set according to this table. An important parameter is TFCONJFILENAME. The files created in harktool3 from impulse responses of a robot are used. Among the seven modules in MAINLOOP (iterator), an important module is GHDSS. Its parameter list is shown in Table 13.16.

13.6.3 Off-line sound source separation (with postprocessing by HRLE)

A sample in which noise estimation with HRLE is performed a postprocessing for separated sounds and noise elimination processing is performed based on the estimated values is introduced here. Since input sounds are files,

Table 13.15: Parameter list of **MAINLOOP**

Parameter name	Type	Set value	Unit	Description
LENGTH	int	512	[pt]	FFT length
ADVANCE	int	160	[pt]	Shift length
SAMPLINGRATE	int	16000	[Hz]	Sampling frequency
TFCONJFILENAME	int	ARG1		TFCONJ file name
DOWHILE	bool			Blank

Table 13.16: Parameter list of GHDSS

Parameter name	Type	Set value	Unit	Description
LENGTH	subnetparam	LENGTH	[pt]	FFT length
ADVANCE	subnetparam	ADVANCE	[pt]	Shift length
SAMPLINGRATE	subnetparam	SAMPRINGRATE	[Hz]	Sampling frequency
LOWERBOUNDFREQUENCY	int	0	[Hz]	Lower limit frequency of processing object
HIGHERBOUNDFREQUENCY	int	8000	[Hz]	Upper limit frequency of processing object
TFCONJ	string	DATABASE		Mode with measured value
TFCONJFILENAME	subnetparam	TFCONJFILENAME		A TFCONJ file name
FIXEDNOISE	bool	false		No directivity fixed noise source
INITWFILENAME	string			Initial value of W is not given
SSMETHOD	string	ADAPTIVE		Adaptation calculation
SSSCAL	float	1.0		
NOISEFLOOR	float	0		
LC_CONST	string	FULL		Update all elements of separation matrix
LC_METHOD	string	ADAPTIVE		Adaptation calculation
UPDATE_METHOD_TF_CONJ	string	POS		Update TFCONJ for each direction
UPDATE_METHOD_W	string	ID		Update W for each sound source ID
UPDATE_ACCEPT_ANGLE	float	0.1		Permissive angle error
EXPORTW	bool	false		Do not output W
UPDATE	string	STEP		Update for each update step

even the users who do not have a multi-channel AD/DA can confirm while executing sound source localization processing. Execute demoOfflineHRLE.sh in the Separation directory. An execution example is shown in Figure 13.23 execute example. After the execution, the sound sources are separated and sounds in which postprocessing is adapted separated sounds are generated. When separation cannot be performed well, check the following items.

1. Check if the ghdss.dat files are in the ../config directory. They are impulse response files of a virtual robot. Sound source localization results are not indicated without these files. Check if the sample2spdat files are in the ../data directory. These files are simulation synthetic sounds of the 8ch sound recording audio waveforms to which impulse responses from the -60 and 60 degrees direction are superimposed. They are interpreted as sounds obtained in the case sounds are given to a virtual robot from each of the -60 and 60 degrees direction simultaneously. If these files are not present, it means there are no inputs for sound source separation and therefore sound source separation cannot be performed.

Fifteen modules are included in this sample. There are three modules in MAINLOOP (iterator) and twelve modules in MAIN (subnet). MAIN (subnet) and MAINLOOP (iterator) are shown in Figures 13.24 and 13.25. The audio waveforms read from the files in the AudioStreamFromWavemodule are analyzed in MultiFFT, separated in GHDSS, postprocessed, synthesized in Synthesizeand the audio waveforms are saved in SaveRawPCM. The postprocessing is realized by combinations of HRLE, EstimateLeak, CalcSpecAddPower, CalcSpecSubGainand SpectralGainFilter. Interference from a non-purpose sound is estimated from nondirectional noise and the sound sources detected and spectral levels for each band are adjusted. ConstantLocalization, which outputs the information that sound sources are localized from the designated direction constantly, is used for sound source localization. Changing this module to LocalizeMUSICin accordance with the sample of Localization and enables to perform sound source localization at the same time. The property of the MAINLOOP module in MAIN (subnet) has five parameters. Its parameter list is shown in Table 13.17. Each parameter value is set according to this table. LENGTH, ADVANCE, SAMPLINGRATE and TFCONJFILENAME are provided from command line arguments of demoOfflineHRLE.n for each of them. An important parameter is TFCONJFILENAME. The files created in harktool3 from impulse responses of a robot are used. Among the 12 modules in MAINLOOP (iterator), an important module is GHDSS, HRLE. Indicate a parameter to table 13.18, 13.19.

13.7 Acoustic feature extraction network sample

There are six kinds of acoustic feature extraction network samples as shown in Table 13.20. The left column indicates network files, the middle column indicates shell script files to operate the networks and the right column

Figure 13.23: Execution example of demoOfflineHRLE.sh.

```
> demoOfflineHRLE.sh
UINodeRepository::Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN
TF = 1,INITW = 0,FixedNoise = 0
SSMethod = 0LCCONST = 0LCMETHOD = 0
NF =257NSrc = 72NMic = 8 Data =
Location =
Robot =
ArrayType =
Creator =
Memo =
```

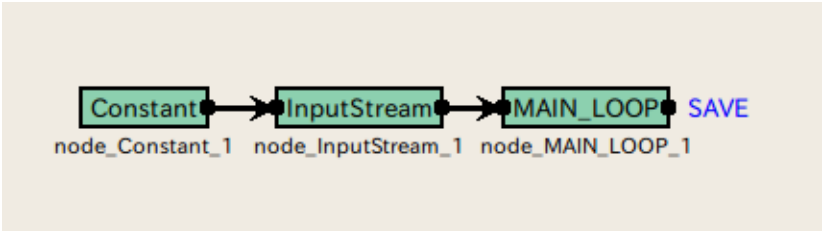


Figure 13.24: MAIN (subnet)

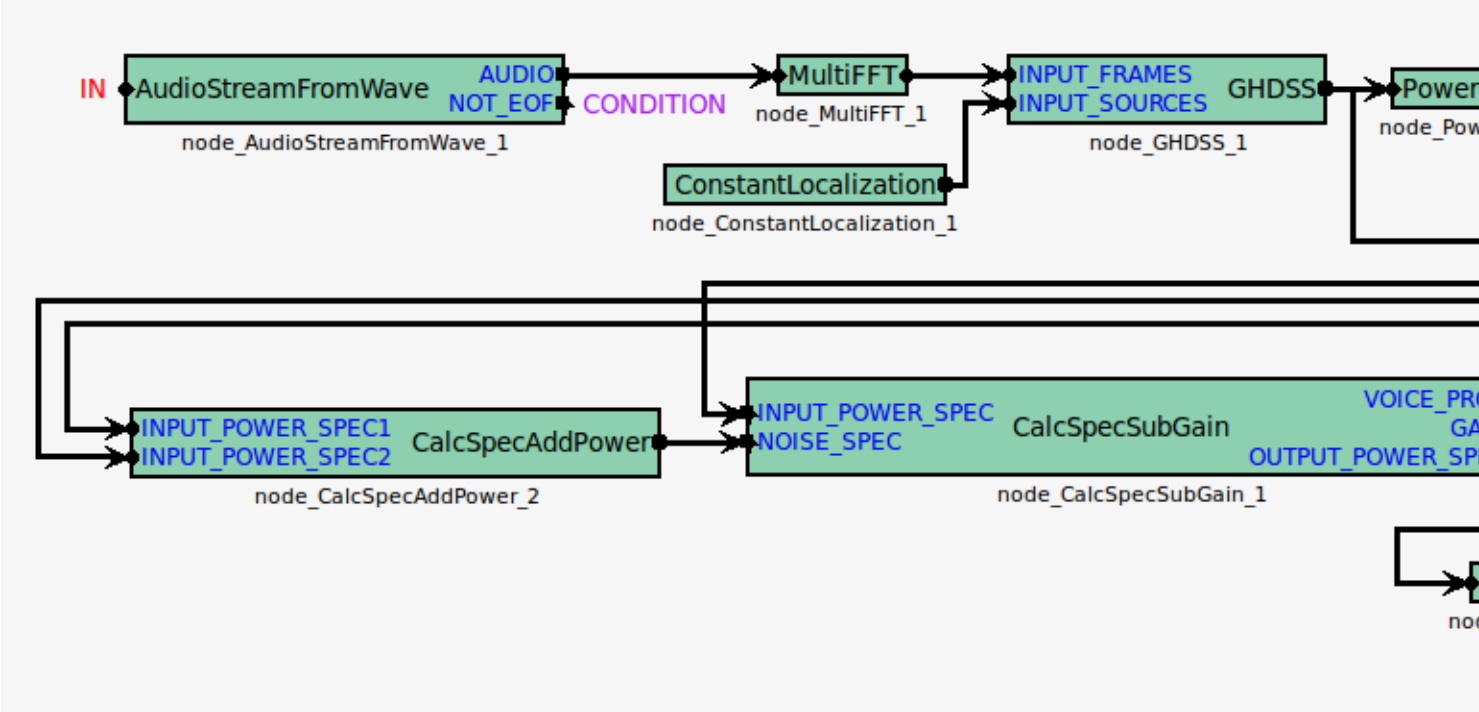


Figure 13.25: MAINLOOP (iterator)

Table 13.17: Parameter list of **MAINLOOP**

Parameter name	Type	Set value	Unit	Description
LENGTH	subnetparam	int:ARG2	[pt]	FFT length
ADVANCE	subnetparam	int:ARG3	[pt]	Shift length
SAMPLINGRATE	subnetparam	int:ARG4	[Hz]	Sampling frequency
TFCONJFILENAME	subnetparam	ARG5		TFCONJ file name
DOWHILE	bool			Blank

Table 13.18: Parameter list of GHDSS

Parameter name	Type	Set value	Unit	Description
LENGTH	subnetparam	LENGTH	[pt]	FFT length
ADVANCE	subnetparam	ADVANCE	[pt]	Shift length
SAMPLINGRATE	subnetparam	SAMPRINGRATE	[Hz]	Sampling frequency
LOWERBOUNDFREQUENCY	int	0	[Hz]	Lower limit frequency of processing object
HIGHERBOUNDFREQUENCY	int	8000	[Hz]	Upper limit frequency of processing object
TFCONJ	string	DATABASE		Mode with measured value
TFCONJFILENAME	subnetparam	TFCONJFILENAME		TFCONJ file name
FIXEDNOISE	bool	false		No directivity-fixed noise source
INITWFILENAME	string			Do not give initial value of W
SSMETHOD	string	ADAPTIVE		Adaptation calculation
SSSCAL	float	1.0		
NOISEFLOOR	float	0		
LCCONST	string	FULL		Update all elements of separation matrix
LCMETHOD	string	ADAPTIVE		Adaptation calculation
UPDATEMETHODTFCONJ	string	POS		Update TFCONJ for each direction
UPDATEMETHODW	string	ID		Update W for each sound source ID
UPDATEACCEPTANGLE	float	0.1		Permissive angle error
EXPORTW	bool	false		Do not output W
UPDATE	string	STEP		Update for each update step

Table 13.19: Parameter list of HRLE

Parameter name	Type	Set value	Unit	Description
LX	float	0.24	[pt]	
TIMECONSTANT	float	16000	[pt]	
NUMBIN	float	2000		
MINLEVEL	float	-200		
STEPLEVEL	float	0.2		

indicates processing contents. Since the acoustic feature extraction network samples are intended for description

Table 13.20: Acoustic feature extraction network list.

Network file name	Network execution script	Processing content
demo1.n	demo1.sh	MSLS extraction (without delta term, without power term)
demo2.n	demo2.sh	MSLS extraction (with delta term, without power term)
demo3.n	demo3.sh	MSLS extraction (without delta term, with power term)
demo4.n	demo4.sh	MSLS extraction (with delta term, with power term, with delta power term)
demo5.n	demo5.sh	MSLS extraction (with delta term, without power term, with delta power ter
demo6.n	demo6.sh	MSLS extraction (with pre-emphasis, with mean subtraction, with delta term without power term, with delta power term)

especially for features extraction, all the samples are of off-line data processing. Adding configuration of this extraction module to a sound recording network sample enbles to make it on-line.

13.7.1 MSLS feature extraction (without delta term without power term)

An MSLS extraction sample is introduced first. Since input sounds are files, even the users who do not have a multi-channel AD/DA can confirm while executing sound source localization processing. Execute demo1.sh in the FeatureExtraction directory. An execution example is shown in Figure 13.26. After the execution, a file named

Figure 13.26: Execution example of demo1.sh.

```
> demo1.sh
UINodeRepository:: Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN
```

MFBANK130.spec is generated. This file stores little endian 13 dimensional vector sequence expressed in the 32 bit floating-point number format. When separation cannot be performed well, check the following items.

1. Check if the101b001.wav files are in the ../data directory.

Twelve modules are included in this sample. There are three modules in MAINLOOP (iterator) and nine modules in MAIN (subnet). MAIN (subnet) and MAINLOOP (iterator) are shown in Figures 13.27 and 13.28. As an outline of the processing, it is simple network configuration in which acoustic features are calculated in MSLSExtractionwith the audio waveforms collected in the AudioStreamFromWavemodule and are written in SaveFeatures. Since MSLSExtractionrequires the outputs of the mel-scale filter bank and power spectra for calculation of MSLS, the collected audio waveforms are analyzed by MultiFFTand their data type are converted by MatrixToMapand PowerCalcForMap, and then processing to obtain outputs of the mel-scale filter bank is performed by MelFilterBank. MSLSExtractionreserves a storing region for the δ MSLS coefficient other than the MSLS coefficient and outputs vectors that are double of the values specified in the FBANKCOUNT property of MSLSExtractionas a feature (zero is in the storing region for theδ MSLS coefficient). Therefore, it is necessary to delete the δ MSLS coefficient domain, which is unnecessary here. Use FeatureRemoverto delete it. SaveFeaturessaves the input FEATURE. If there are no inputs for SOURCES, an error occurs, and therefore appropriate localization information is generated in ConstantLocalizationand sent to SOURCES. The property of the MAINLOOP module in MAIN

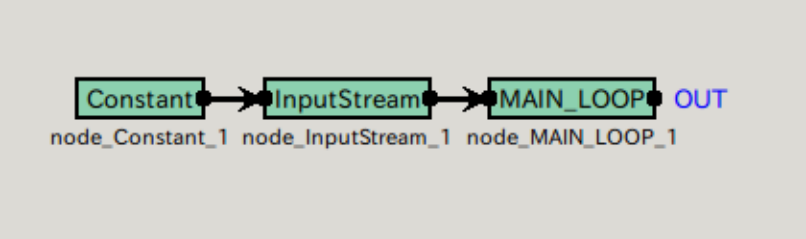


Figure 13.27: MAIN (subnet)

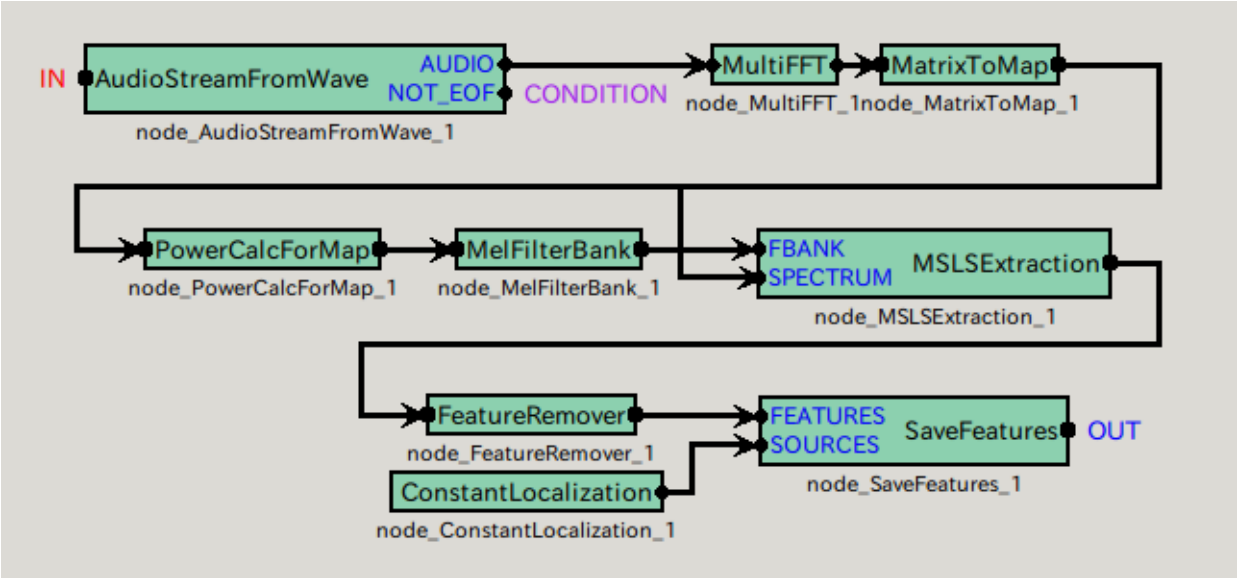


Figure 13.28: MAINLOOP (iterator)

(subnet) has five parameters. Its parameter list is shown in Table 13.21. Each parameter value is set according to this table. Among the nine modules in MAINLOOP (iterator), an important module is MSLSExtraction. Its

Table 13.21: Parameter list of **MAINLOOP**

Parameter name	Type	Set value	Unit	Description
LENGTH	subnetparam	`int`:ARG2	[pt]	FFT length
ADVANCE	subnetparam	`int`:ARG3	[pt]	Shift length
SAMPLINGRATE	subnetparam	`int`:ARG4	[Hz]	Sampling frequency
FBANKCOUNT	subnetparam	`int`:ARG5	[ch]	Number of filter banks
DOWHILE	`bool`			Blank

parameter lis is shown in Table 13.22.

Table 13.22: Parameter list of MSLSExtraction

Parameter name	Type	Set value	Unit	Description
FBANKCOUNT	subnetparam	FBANKCOUNT	[ch]	Number of filter banks
NORMALIZEMODE	`string`	Cepstral		Normalization
USEPOWER	`bool`	`false`		Without power term

13.7.2 MSLS feature extraction (with delta term, without power term)

An MSLS extraction sample is introduced here. Since input sounds are files, even the users who do not have a multi-channel AD/DA can confirm while executing sound source localization processing. Execute demo2.sh in the FeatureExtraction directory. An execution example is shown in Figure 13.29. After the execution, a file named MFBANK260.spec is generated. This file stores little endian 26 dimensional vector sequence expressed in the 32 bit floating-point number format. When separation cannot be performed well, check the following items.

1. Check if the101b001.wav files are in the ../data directory.

Twelve modules are included in this sample. There are three modules in MAINLOOP (iterator) and nine modules in MAIN (subnet). MAIN (subnet) and MAINLOOP (iterator) are shown in 13.30 and 13.31. As an outline of the processing, it is simple network configuration in which acoustic features are calculated in MSLSExtractionwith the audio waveforms collected in the AudioStreamFromWavemodule and are written in SaveFeatures. Since MSLSExtractionrequires the outputs of the mel-scale filter bank and power spectra for calculation of MSLS, the collected audio waveforms are analyzed by MultiFFTand their data type are converted by MatrixToMapand

Figure 13.29: Execution example of demo2.sh.

```
> demo2.sh
UINodeRepository:: Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN
```

PowerCalcForMap, and then processing to obtain outputs of the mel-scale filter bank is performed by MelFilterBank. MSLSExtractionreserves a storing region for the δ MSLS coefficient other than the MSLS coefficient and outputs vectors that are double of the values specified in the FBANKCOUNT property of MSLSExtractionas a feature. zero is in the storing region for theδ MSLS coefficient. The δ MSLS coefficient is calculated and stored with Delta. SaveFeaturessaves the input FEATURE. If there are no inputs for SOURCES, an error occurs, and therefore appropriate localization information is generated in ConstantLocalizationand sent to SOURCES. The

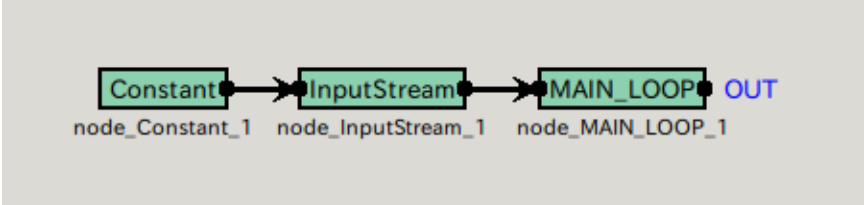


Figure 13.30: MAIN (subnet)

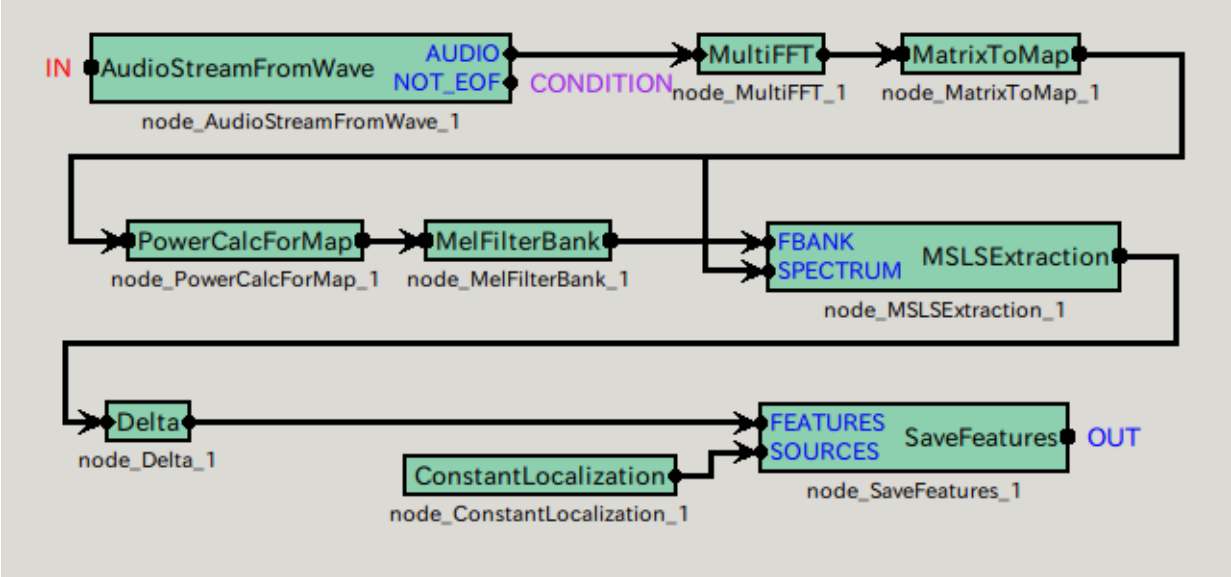


Figure 13.31: MAINLOOP (iterator)

property of the MAINLOOP module in MAIN (subnet) has five parameters. Its parameter list is shown in Table 13.23. Each parameter value is set according to this table. Among the nine modules in MAINLOOP (iterator), an important module is Delta. Its parameter list is shown in Table 13.24.

13.7.3 MSLS feature extraction (without delta term, with power term)

An MSLS extraction sample is introduced here. Since input sounds are files, even the users who do not have a multi-channel AD/DA can confirm while executing sound source localization processing. Execute demo3.sh in the

Table 13.23: Parameter list of **MAINLOOP**

Parameter name	Type	Set value	Unit	Description
LENGTH	subnetparam	int:ARG2	[pt]	FFT length
ADVANCE	subnetparam	int:ARG3	[pt]	Shift length
SAMPLINGRATE	subnetparam	int:ARG4	[Hz]	Sampling frequency
FBANKCOUNT	subnetparam	int:ARG5	[ch]	Number of filter banks
DOWHILE	bool			Blank

Table 13.24: Parameter list of Delta

Parameter name	Type	Set value	Unit	Description
FBANKCOUNT	subnetparam	FBANKCOUNT	[ch]	Number of filter banks

FeatureExtraction directory. An execution example is shown in Figure 13.32. After the execution, a file named

Figure 13.32: Execution example of demo3.sh.

```
> demo3.sh
UINodeRepository:: Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN
```

MFBANK140.spec is generated. This file stores little endian 14 dimensional vector sequence expressed in the 32 bit floating-point number format. When separation cannot be performed well, check the following items.

1. Check if the101b001.wav files are in the ../data directory.

Twelve modules are included in this sample. There are three modules in MAINLOOP (iterator) and nine modules in MAIN (subnet). MAIN (subnet) and MAINLOOP (iterator) are shown in 13.33 and 13.34 As an outline of the processing, it is simple network configuration in which acoustic features are calculated in MSLSExtractionwith the audio waveforms collected in the AudioStreamFromWavemodule and are written in SaveFeatures. Since MSLSExtractionrequires the outputs of the mel-scale filter bank and power spectra for calculation of MSLS, the collected audio waveforms are analyzed by MultiFFTand their data type are converted by MatrixToMapand PowerCalcForMap, and then processing to obtain outputs of the mel-scale filter bank is performed by MelFilterBank. Here, the USEPOWER property of MSLSExtractionis set to `true`and to output the power term at the same time. MSLSExtractionreserves a storing region for the δ MSLS coefficient other than the MSLS coefficient and outputs vectors as a feature (zero is in the storing region for theδ MSLS coefficient). Since the USEPOWER property is set to `true`, a storing region of δ MSLS and the delta power term is secured for the δ coefficient. Therefore, vectors that are double of the values specified in the FBANKCOUNT property of MSLSExtraction+1 are output as a feature. Outputs other than the necessary MSLS coefficient and item power term are deleted in FeatureRemover. SaveFeaturessaves the input FEATURE. If there are no inputs for SOURCES, an error occurs, and therefore appropriate localization information is generated in ConstantLocalizationand sent to SOURCES.

The property of the MAINLOOP module in MAIN (subnet) has five parameters. Its parameter list is shown in Table 13.25. Each parameter value is set according to this table. Among the nine modules in MAINLOOP (iterator), an important module is MSLSExtraction. Its parameter list is shown in Table 13.26.

13.7.4 MSLS feature extraction (with delta term, with power term, with delta power term)

An MSLS extraction sample is introduced here. Since input sounds are files, even the users who do not have a multi-channel AD/DA can confirm while executing sound source localization processing. Execute demo4.sh in the FeatureExtraction directory. An execution example is shown in Figure 13.35. After the execution, a file named MFBANK280.spec is generated. This file stores little endian 28 dimensional vector sequence expressed in the 32 bit floating-point number format. When separation cannot be performed well, check the following items.

1. Check if the101b001.wav files are in the ../data directory.

Twelve modules are included in this sample is dozen. There are three modules in MAINLOOP (iterator) and nine modules in MAIN (subnet). MAIN (subnet) and MAINLOOP (iterator) are shown in 13.36 and 13.37. As an

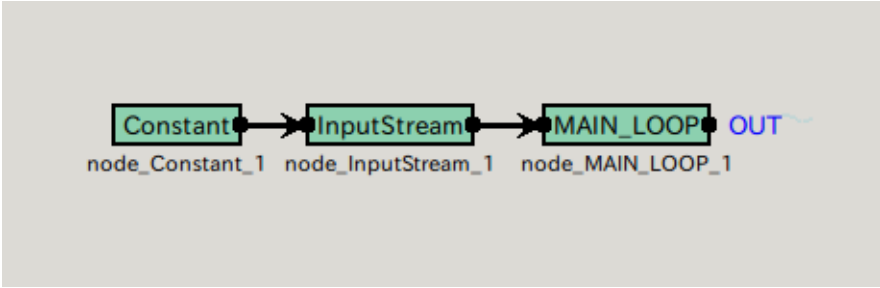


Figure 13.33: MAIN (subnet)

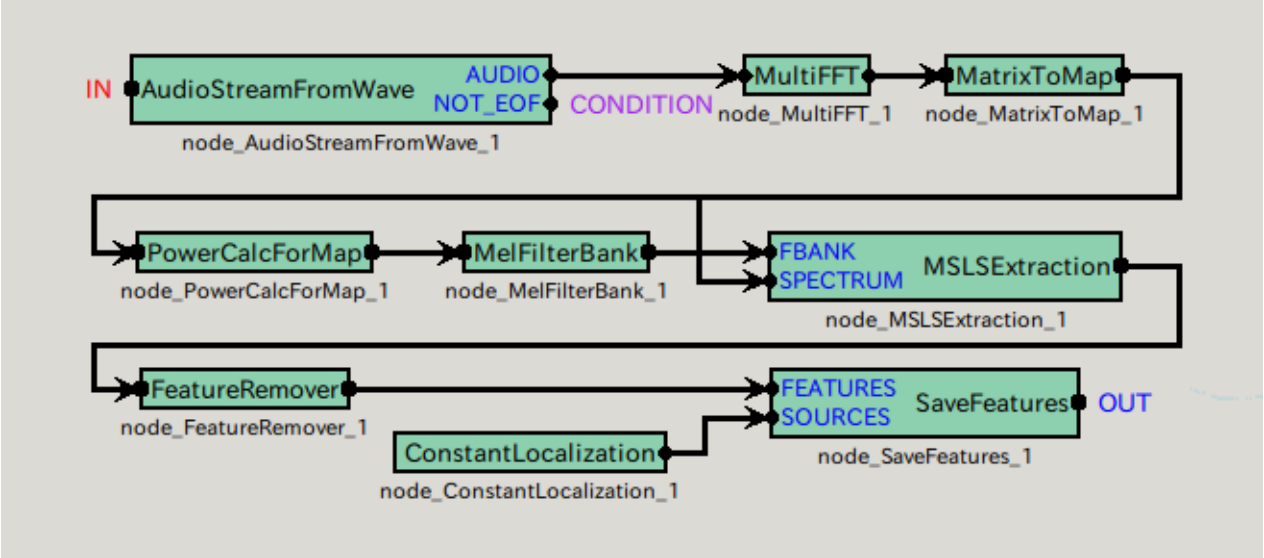


Figure 13.34: MAINLOOP (iterator)

Table 13.25: Parameter list of **MAINLOOP**

Parameter name	Type	Set value	Unit	Description
LENGTH	subnetparam	`int`:ARG2	[pt]	FFT length
ADVANCE	subnetparam	`int`:ARG3	[pt]	Shift length
SAMPLINGRATE	subnetparam	`int`:ARG4	[Hz]	Sampling frequency
FBANKCOUNT	subnetparam	`int`:ARG5	[ch]	Number of filter banks
DOWHILE	`bool`			Blank

Table 13.26: Parameter list of MSLSExtraction

Parameter name	Type	Set value	Unit	Description
FBANKCOUNT	subnetparam	FBANKCOUNT	[ch]	Number of filter banks
NORMALIZEMODE	`string`	Cepstral		Normalization
USEPOWER	`bool`	`true`		With power term

Figure 13.35: Execution example of demo4.sh.

```
> demo4.sh
UINodeRepository:: Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN
```

outline of the processing, it is simple network configuration in which acoustic features are calculated in MSLSExtractionwith the audio waveforms collected in the AudioStreamFromWavemodule and are written in SaveFeatures. Since MSLSExtractionrequires the outputs of the mel-scale filter bank and power spectra for calculation of MSLS, the collected audio waveforms are analyzed by MultiFFTand their data type are converted by MatrixToMapand PowerCalcForMap, and then processing to obtain outputs of the mel-scale filter bank is performed by MelFilterBank. Here, the USEPOWER property of MSLSExtractionis set to `true`and to output the power term at the same time. MSLSExtractionreserves a storing region for the δ MSLS coefficient other than the MSLS coefficient and outputs vectors as a feature (zero is in the storing region for theδ MSLS coefficient). Since the USEPOWER property is set to `true`, a storing region of δ MSLS and the delta power term is secured for the δ coefficient. Therefore, vectors that are double of the values specified in the FBANKCOUNT property of MSLSExtraction+1 are output as a feature. The δ MSLS coefficient and delta power term are calculated and stored with Delta. SaveFeaturessaves the input FEATURE. If there are no inputs for SOURCES, an error occurs, and therefore appropriate localization information is generated in ConstantLocalizationand sent to SOURCES. The property of

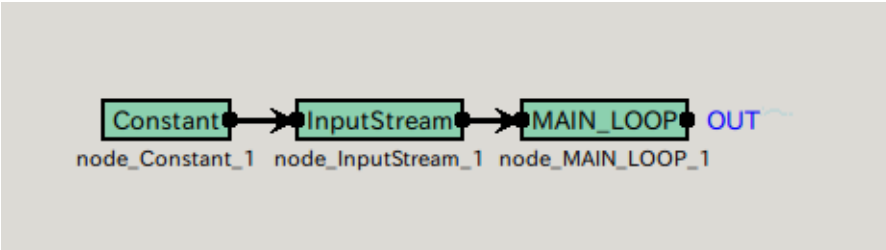


Figure 13.36: MAIN (subnet)

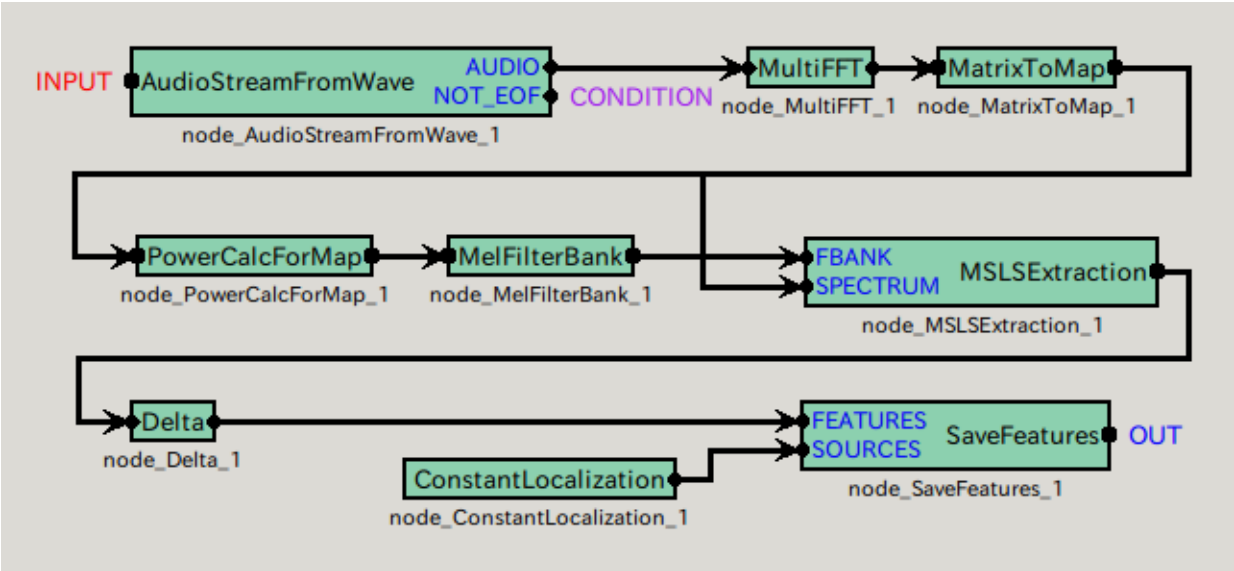


Figure 13.37: MAINLOOP (iterator)

the MAINLOOP module in MAIN (subnet) has six parameters. Its parameter list is shown in Table 13.27. Each parameter value is set according to this table. Among the nine modules in MAINLOOP (iterator), important

Table 13.27: Parameter list of **MAINLOOP**

Parameter name	Type	Set value	Unit	Description
LENGTH	subnetparam	`int`:ARG2	[pt]	FFT length
ADVANCE	subnetparam	`int`:ARG3	[pt]	Shift length
SAMPLINGRATE	subnetparam	`int`:ARG4	[Hz]	Sampling frequency
FBANKCOUNT	subnetparam	`int`:ARG5	[ch]	Number of filter banks
FBANKCOUNT1	subnetparam	`int`:ARG6	[ch]	Number of filter banks +1
DOWHILE	`bool`			Blank

modules are MSLSExtractionand Delta. Their parameter lists are shown in Table 13.28 and 13.29.

Table 13.28: Parameter list of MSLSExtraction

Parameter name	Type	Set value	Unit	Description
FBANKCOUNT	subnetparam	FBANKCOUNT	[ch]	Number of filter banks
NORMALIZEMODE	`string`	Cepstral		Normalization
USEPOWER	`bool`	`true`		Power term

Table 13.29: Parameter list of Delta

Parameter name	Type	Set value	Unit	Description
FBANKCOUNT1	subnetparam	FBANKCOUNT1		Number of filter banks +1

13.7.5 MSLS feature extraction (with delta term, without power term, with delta power term)

An MSLS extraction sample is introduced here. Since input sounds are files, even the users who do not have a multi-channel AD/DA can confirm while executing sound source localization processing. Execute demo5.sh in the FeatureExtraction directory. An execution example is shown in Figure 13.38. After the execution, a file named

Figure 13.38: Execution example of demo5.sh.

```
> demo5.sh
UINodeRepository:: Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN
```

MFBANK270.spec is generated. This file stores little endian 27 dimensional vector sequence expressed in the 32 bit floating-point number format. When feature extraction cannot be performed well, check the following items.

1. Check if the101b001.wav files are in the ../data directory.

Thirteen modules are included in this sample. There are three modules in MAINLOOP (iterator) and ten modules in MAIN (subnet). MAIN (subnet) and MAINLOOP (iterator) are shown in Figures 13.39 and 13.40 As an outline of the processing, it is simple network configuration in which acoustic features are calculated in MSLSExtractionwith the audio waveforms collected in the AudioStreamFromWavemodule and are written in SaveFeatures. Since MSLSExtractionrequires the outputs of the mel-scale filter bank and power spectra for calculation of MSLS, the collected audio waveforms are analyzed by MultiFFTand their data type are converted by MatrixToMapand PowerCalcForMap, and then processing to obtain outputs of the mel-scale filter bank is performed by MelFilterBank. MSLSExtractionreserves a storing region for the δ MSLS coefficient other than the MSLS coefficient and outputs vectors as a feature (zero is in the storing region for theδ MSLS coefficient). Since the USEPOWER property is set to `true`, a storing region of δ MSLS and the delta power term is secured for the δ coefficient.

Therefore, vectors that are double of the values specified in the FBANKCOUNT property of MSLSExtraction+1 are output as a feature. The δ MSLS coefficient and delta power term are calculated and stored with Delta. Since necessary coefficients are the MSLS coefficient and δ MSLS coefficient and delta power term, it is necessary to delete unnecessary power terms. Use FeatureRemoverto delete them. SaveFeaturessaves the input FEATURE. If there are no inputs for SOURCES, an error occurs, and therefore appropriate localization information is generated in ConstantLocalizationand sent to SOURCES. The property of a MAINLOOP module in MAIN (subnet) has six parameters. Its parameter list is shown in Table 13.30. Each parameter value is set according to this table. Among the ten modules in MAINLOOP (iterator), important modules are MSLSExtractionand Deltaand FeatureRemover. Their parameter lists are shown in Table **??**.

13.7.6 MSLS feature extraction (with pre-emphasis, with mean subtration, with delta term, without power term, with delta power term)

An MSLS extraction sample is introduced here. Since input sounds are files, even the users who do not have a multi-channel AD/DA can confirm while executing sound source localization processing. Execute demo6.sh in the

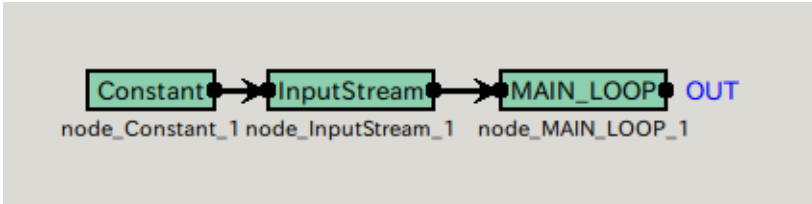


Figure 13.39: MAIN (subnet)

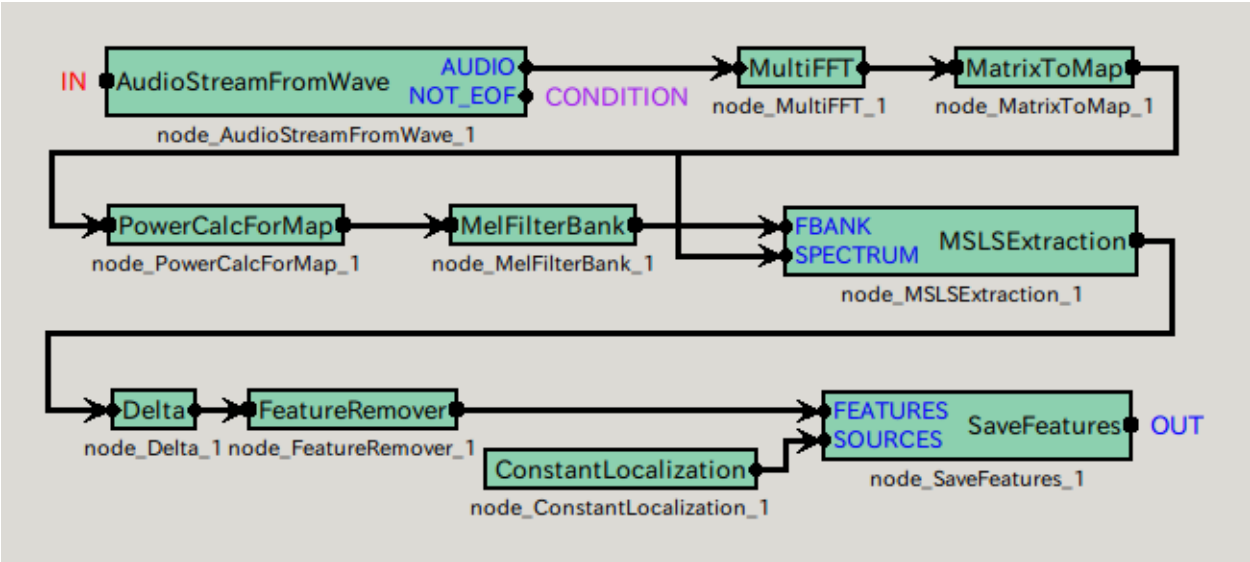


Figure 13.40: MAINLOOP (iterator)

Table 13.30: Parameter list of **MAINLOOP**

Parameter name	Type	Set value	Unit	Description
LENGTH	subnetparam	int:ARG2	[pt]	FFT length
ADVANCE	subnetparam	int:ARG3	[pt]	Shift length
SAMPLINGRATE	subnetparam	int:ARG4	[Hz]	Sampling frequency
FBANKCOUNT	subnetparam	int:ARG5	[ch]	Number filter banks
FBANKCOUNT1	subnetparam	int:ARG6	[ch]	Number of silter banks +1
DOWHILE	bool			Blank

Table 13.31: Parameter list of MSLSExtraction

Parameter name	Type	Set value	Unit	Description
FBANKCOUNT	subnetparam	FBANKCOUNT	[ch]	Number of filter banks
NORMALIZEMODE	string	Cepstral		Normalization
USEPOWER	bool	true		With power term

Table 13.32: Parameter list of Delta

Parameter name	Type	Set value	Unit	Description
FBANKCOUNT1	subnetparam	FBANKCOUNT1		Number of filter banks +1

Table 13.33: Parameter list of FeatureRemover

Parameter name	Type	Set value	Unit	Description
SELECTOR	Object	<Vector<Float> 13 >		Dimension to be deleted

Figure 13.41: Executione example of demo6.sh.

```
> demo6.sh
UINodeRepository:: Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN
```

FeatureExtraction directory. An execution example is shown in Figure 13.41. After the execution, a file named MFBANK270.spec is generated. This file stores little endian 27 dimensional vector sequence expressed in the 32 bit floating-point number format. When feature axtraction cannot be performed well, check the following items.

1. Check if the101b001.wav files are in the ../data directory.

Seventeen modules are included in this sample. There are three modules in MAINLOOP (iterator) and fourteen modules in MAIN (subnet). MAIN (subnet) MAIN (subnet) and MAINLOOP (iterator) are shown in Figures 13.42 and 13.43 As an outline of the processing, it is simple network configuration in which acoustic features are calculated in MSLSExtractionwith the audio waveforms collected in the AudioStreamFromWavemodule and are written in SaveFeatures. Since pre-emphasis is performed for over the time domain, after analyzing audio waveforms in MultiFFT, their type is converted with MatrixToMapand the signals are synthesized by Synthesizeonce. Pre-emphasis is performed for the synthesized waves with PreEmphasis, they are analyzed with MultiFFTonce more, their type is converted with PowerCalcForMapand sent to MSLSExtraction. Since MSLSExtractionrequires the outputs of the mel-scale filter bank and power spectra for calculation of MSLS, the collected audio waveforms are analyzed by MultiFFTand their data type are converted by MatrixToMapand PowerCalcForMap, and then processing to obtain outputs of the mel-scale filter bank is performed by MelFilterBank. MSLSExtractionreserves a storing region for the δ MSLS coefficient other than the MSLS coefficient and outputs vectors as a feature (zero is in the storing region for theδ MSLS coefficient). Since the USEPOWER property is set to `true`, a storing region of δ MSLS and the delta power term is secured for the δ coefficient. herefore, vectors that are double of the values specified in the FBANKCOUNT property of MSLSExtraction+1 are output as a feature. Pssing through SpectralMeanNormalization, which performs mean subtraction, the δ MSLS coefficient and delta power term are calculated and stored with Delta. Since necessary coefficients are the MSLS coefficient and δ MSLS coefficient and delta power term, it is necessary to delete unnecessary power terms. Use FeatureRemoverto delete them. SaveFeaturessaves the input FEATURE. If there are no inputs for SOURCES, an error occurs, and therefore appropriate localization information is generated in ConstantLocalizationand sent to SOURCES. The property

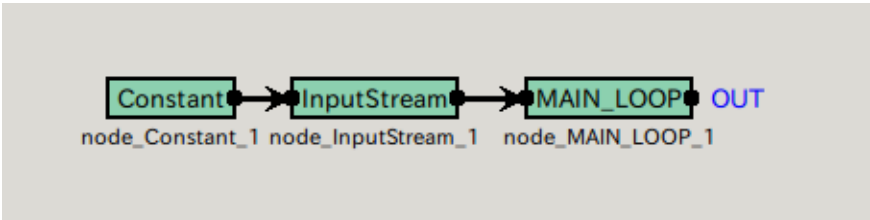


Figure 13.42: MAIN (subnet)

of the MAINLOOP module in MAIN (subnet) has six parameters. Its parameter list is shown in Table 13.34. Each parameter value is set according to this table. Among the fourteen modules in MAINLOOP (iterator), im-

Parameter list of

Table 13.34: **MAINLOOP**

Parameter name	Type	Set value	Unit	Description
LENGTH	subnetparam	`int`:ARG2	[pt]	FFT length
ADVANCE	subnetparam	`int`:ARG3	[pt] A	shift length
SAMPLINGRATE	subnetparam	`int`:ARG4	[Hz]	Sampling frequency
FBANKCOUNT	subnetparam	`int`:ARG5	[ch]	Number of filter banks
FBANKCOUNT1	subnetparam	`int`:ARG6	[ch]	Number of filter banks +1
DOWHILE	`bool`			Blank

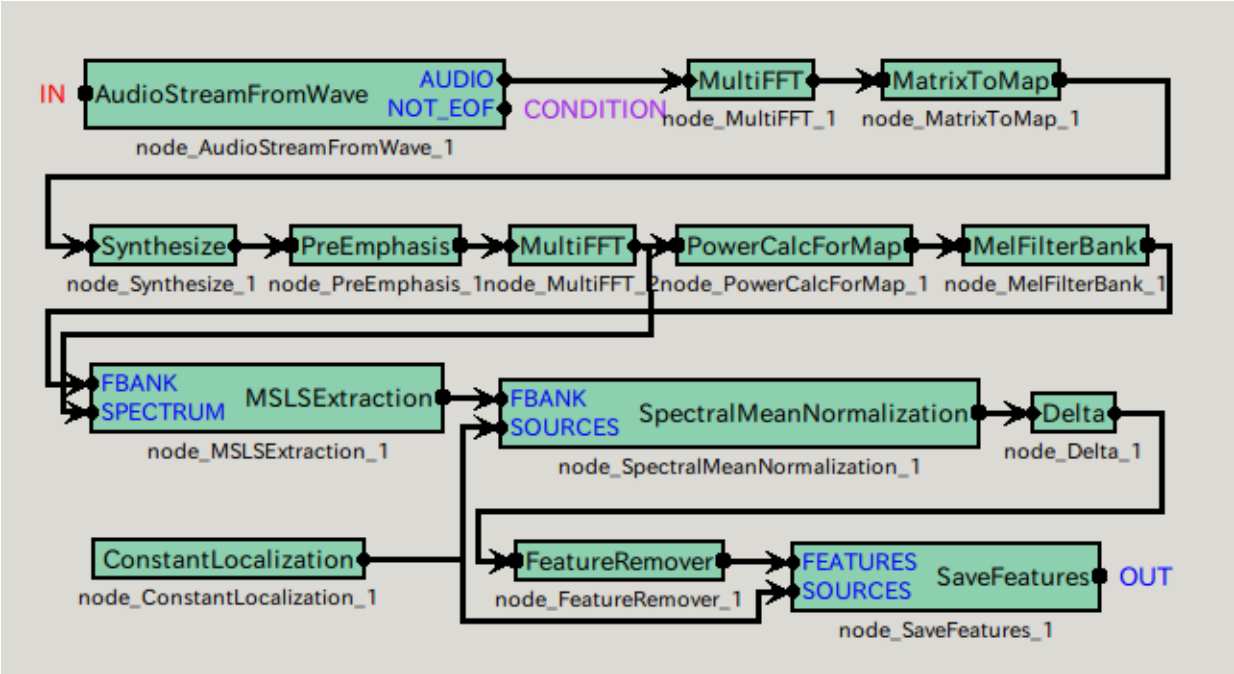


Figure 13.43: MAINLOOP (iterator)

portant modules are PreEmphasis, MSLSExtraction, SpectralMeanNormalization, Deltaand FeatureRemover. Their parameter lists are shown in Tables 13.35,13.36,13.37,13.38 and 13.39.

Table 13.35: Parameter list of PreEmphasis

Parameter name	Type	Set value	Unit	Description
LENGTH	subnetparam	LENGTH	[pt]	FFT length
SAMPLINGRATE	subnetparam	SAMPLINGRATE	[Herz]	Sampling frequency
PREEMCOEFF	`float`	0.97		Pre-emphasis coefficient
INPUTTYPE	`string`	WAV		Time domain processing

Table 13.36: Parameter list of MSLSExtraction

Parameter name	Type	Set value	Unit	Description
FBANKCOUNT	subnetparam	FBANKCOUNT	[ch]	Number of filter banks
NORMALIZEMODE	`string`	Cepstral		Normalization
USEPOWER	`bool`	`true`		With power term

13.8 Speech recognition network sample

There are two kinds of speech recognition network samples as shown in Table 13.40. The left column indicates network files, the middle column indicates shell script files to operate the networks and the right column indicates processing contents. Since the speech recognition network samples are intended for description especially for automatic speech recognition, all the samples are of off-line data processing. Adding configuration of this extraction module to a sound recording network sample enables to make it on-line.

A speech recognition sample with MFCC is introduced first Since input sounds are files, even the users who do not have a multi-channel AD/DA can confirm while executing sound source localization processing. Execute demoMFCC.sh in the Recognition directory. An execution example is shown in Figure 13.44. After the execution, MFCC of separated sounds is transmitted to juliusmft and juliusmft outputs recognition results on the screen. When separation cannot be performed well, check the following items.

1. Check if the101b001.wav files are in the ../data directory.

Table 13.37: Parameter list of SpectralMeanNormalization

Parameter name	Type	Set value	Unit	Description
FBANKCOUNT1	subnetparam	FBANKCOUNT1		Number of filter banks +1

Table 13.38: Parameter list of Delta

Parameter name	Type	Set value	Unit	Description
FBANKCOUNT1	subnetparam	FBANKCOUNT1		Number of filter banks +1

2. Check if juliusMFT is started.

Sixteen modules are included in this sample. There are three modules in MAINLOOP (iterator) and thirteen modules in MAIN (subnet). MAIN (subnet) and MAINLOOP (iterator) are shown in Figures 13.45 and 13.46. For the audio waveforms collected in the AudioStreamFromWavemodule, acoustic features are calculated in MSLSExtractionand the acoustic features are sent to a speech recognition server by socket communication in SpeechRecognitionClient. The property of the MAINLOOP module in MAIN (subnet) has seven parameters. Its parameter list is shown in Table 13.41. Each parameter value is set according to this table.

13.8.1 Separated sound recognition processing based on MFT with MSLS

A separated sound recognition sample based on MFT with MSLS is introduced here. Since input sounds are files, even the users who do not have a multi-channel AD/DA can confirm while executing sound source localization processing. Execute demoMSLS.sh in the Recognition directory. An execution example is shown in Figure 13.47. After the execution, MSLS of separated sounds is transmitted to juliusmft and juliusmft outputs recognition results on the screen. When separation cannot be performed well, check the following items.

1. Check if the101b001.wav files are in the ../data directory.

2. Check if Julius is started.

Twenty five modules are included in this sample. There are three modules in MAINLOOP (iterator) and twenty-two modules in MAIN (subnet). MAIN (subnet) and MAINLOOP (iterator) are shown in 13.48 and 13.49. For the audio waveforms collected in the AudioStreamFromWavemodule, acoustic features are calculated in MSLSExtraction, the missing feature mask is generated in MFMGeneration, and the acoustic features and missing feature mask are sent to a speech recognition server by socket communication in SpeechRecognitionClient.

Table 13.39: Parameter list of FeatureRemover

Parameter name	Type	Set value	Unit	Description
SELECTOR	Object	<Vector<Float> 13 >		Dimension to be deleted

Table 13.40: Speech recognition network list.

Network file name	network execution script	processing content
demoMFCC.n	demoMFCC.sh	Separated sound recognition processing based on MFT with the MFCC featu
demoMSLS.n	demoMSLS.sh	Separated sound recognition processing based on MFT with the MSLS featur

Figure 13.44: Execution example of demoMFCC.sh.

```
> demoMFCC.sh
UINodeRepository::Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN
```

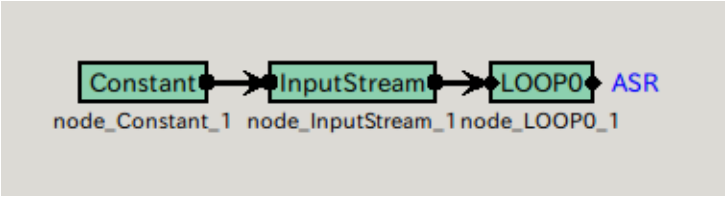


Figure 13.45: MAIN (subnet)

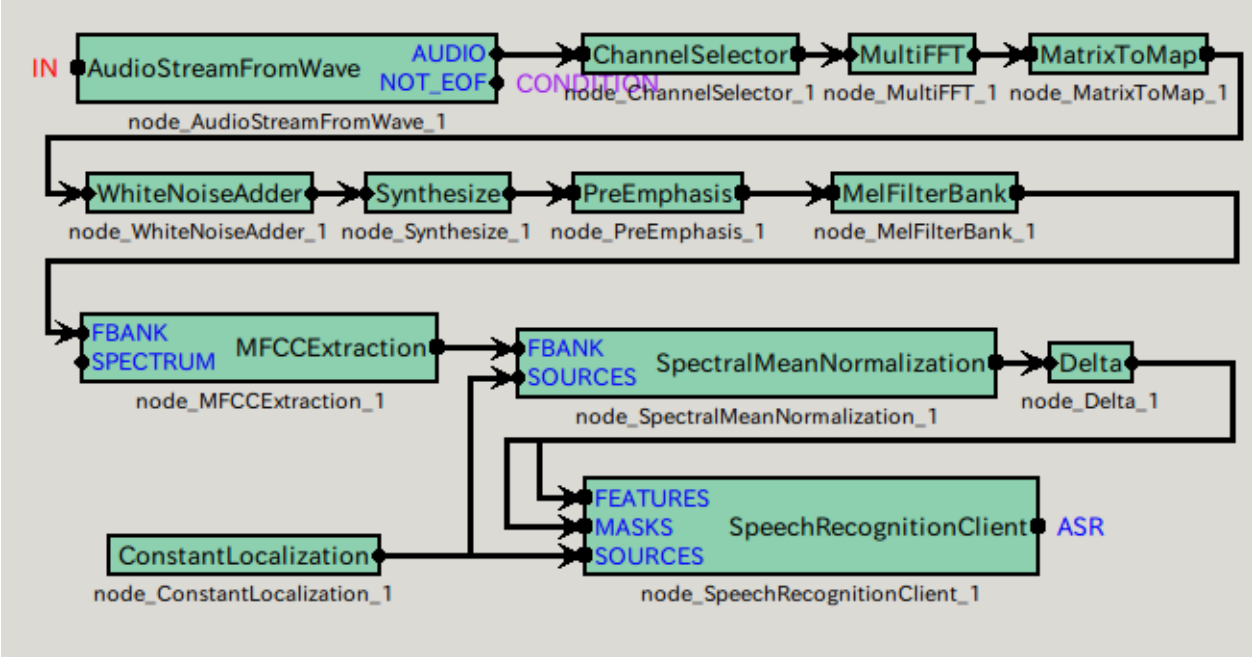


Figure 13.46: MAINLOOP (iterator)

Table 13.41: Parameter list of **MAINLOOP**

Parameter name	Type	Set value	Unit	Description
LENGTH	subnetparam	int:ARG2	[pt]	FFT length
ADVANCE	subnetparam	int:ARG3	[pt]	Shift length
SAMPLINGRATE	subnetparam	int:ARG4	[Hz]	Sampling frequency
FBANKCOUNT	subnetparam	int:ARG5	[ch]	Number of filter banks
NUMCEPS	subnetparam	int:ARG6		Lifteing dimension
WNLEVEL	subnetparam	int:ARG7		Amplitude of white noise
DOWHILE	bool			Blank

Figure 13.47: Execution example of demoMSLS.sh.

```
> demoMSLS.sh
UINodeRepository::Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network : MAIN
```

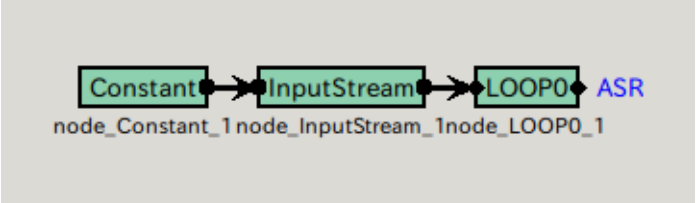


Figure 13.48: MAIN (subnet)

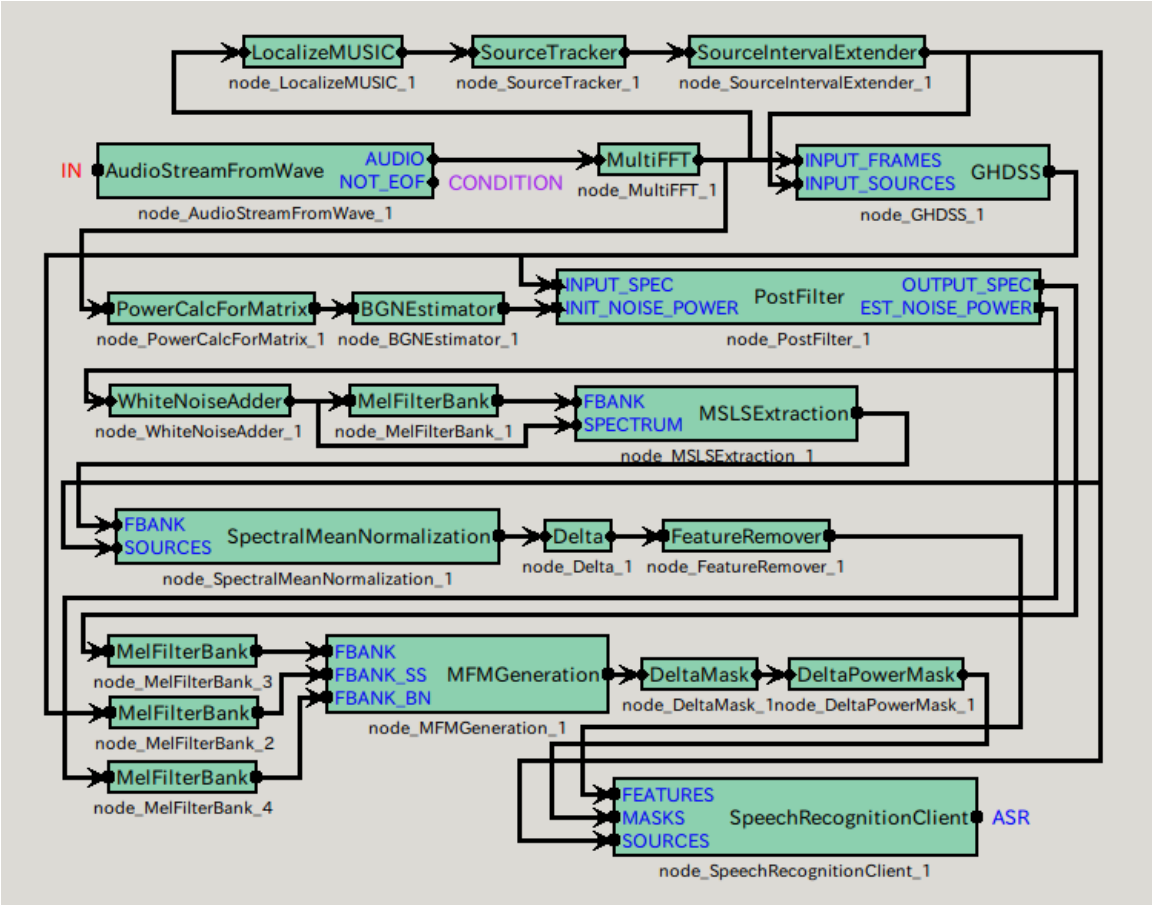


Figure 13.49: MAINLOOP (iterator)