HARK Cookbook
Version 2.3.1. (Revision: 8682)

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

Contents

Chapter 1

Introduction

This document describes frequently occurring problems and their solutions. We call each pair, consisting of a problem and its solution, a "recipe", analogous to a recipe for cooking.

Chapter 2: Learning HARK
This chapter includes recipes for those beginning HARK. These recipes describe methods of recording sounds, and localizing, separating, and recognizing. If this is the first time you are using HARK, we recommend that you read the recipes in this chapter.

Chapter 3: Something is Wrong!
This chapter describes troubleshooting common problems including installation and recording. It also includes debugging recipes. If you have trouble, check this chapter.

Chapter 4: Microphone Array
This chapter includes recipes for designing microphone arrays, including, for example, the number and kinds of microphone to use and how to install a microphone in a robot.

Chapter 5: Input Data Generation
This chapter includes recipes for generating input data for HARK by recording and simulation. These include, for example, multichannel recording, impulse response measurements, and multichannel sound generation by simulation.

Chapter 6: Acoustic and Language Models
The speech recognizer Julius , which is supported by HARK, requires an acoustic model and a language model. This chapter describes how to build these models.

Chapter 7: FlowDesigner
In HARK, you build a robot audition system using FlowDesigner middleware. FlowDesigner includes the placement and connection of nodes. This chapter includes tips for using FlowDesigner . See another document for **HARK Designer**, a newly developed GUI for building a network from Ver 1.9.9.

Chapter 8: Sound Source Localization
This chapter includes recipes for sound source localization including localization system building, debugging, and parameter tuning.

Chapter 9: Sound Source Separation
This chapter includes recipes for sound source separation including separation system building, debugging, and parameter tuning.

Chapter 10: Feature Extraction
Speech recognition requires the extraction of features from separated sound. This chapter includes recipes for the introduction to features used for speech recognition and how to extract them. This chapter also includes the recipes of Missing Feature Theory used to select reliable features.

Chapter 11: Speech Recognition
This chapter includes how to make a configuration file for Julius .

Chapter 12: Others
This chapter includes miscellaneous recipes, such as selecting a window function for frequency analysis.

Chapter 13: Advanced recipes
This chapter includes advanced recipes, such as those for adding a new function to HARK and connecting HARK to other systems.

Chapter 14: Sample Networks
This chapter includes sample network files.

Chapter 2

Learning HARK

2.1 Learning sound recording

Problem

I want to record sound using HARK. This is the first time I am using HARK.

Solution

If this is the first time you are using HARK, start recording because this is a basic function and inevitable if you run HARK online. Note that if you just want to record, wios is convenient. see the HARK document or Recording multichannel sound for details.

Recording device :

First, get a recording device. Although you will need a multichannel recording device if you want to localize or separate sounds, you can use a microphone input on you computer if you just want to record. (see the chapter on devices in the HARK document for details).

Hereafter, we will assume that you are useing ALSA device. Run `arecord -l` on your terminal. You will see the list of devices, which looks like:

```
card 0: Intel [HDA Intel], device 0: AD198x Analog [AD198x Analog]
  Subdevices: 2/2
  Subdevice #0: subdevice #0
  Subdevice #1: subdevice #1
card 1: ** [***], device 0: USB Audio [USB Audio]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

First, find the name of your device in the list. If you cannot find it, the OS does not recognize your device. If you find the device, you can specify the *device name* by two numbers, card and subdevice. For example, if you want to use the USB Audio, the card number is 1 and the subdevice number is #0, making the device name `plughw:1,0`. If you have no idea what device you are using, try `plughw:0,0`.

Building a recording network :

Next, build a recording network using FlowDesigner . The topology of the network is the same as shown in Fig.2.1 and 2.2. Although you can leave almost all parameters as their default, it is possible to change the parameters shown in 2.1 by yourself.

Check the specifications of you device to determine the number of channels and the sampling frequency. The device name has been specified in the section above. The number of frames to record depends on how long you want to record. This can be calculated using the equation:

$$Recording\ time = (LENGTH + (number\ of\ frames - 1) * ADVANCE)/SAMPLING\_RATE \tag{2.1}$$

where the uppercase variables are the parameter names of AudioStreamFromMic . For example, if you want to a record for 5 seconds at 16kHz sampling frequency with the default LENGTH and ADVANCE, the number of frames will be 498 because:

$$5[sec] \quad = \quad (512 + (number\ of\ frames - 1) * 160)/16000 \tag{2.2}$$

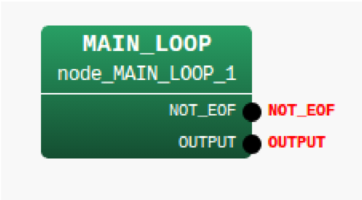


Figure 2.1: MAIN (subnet)

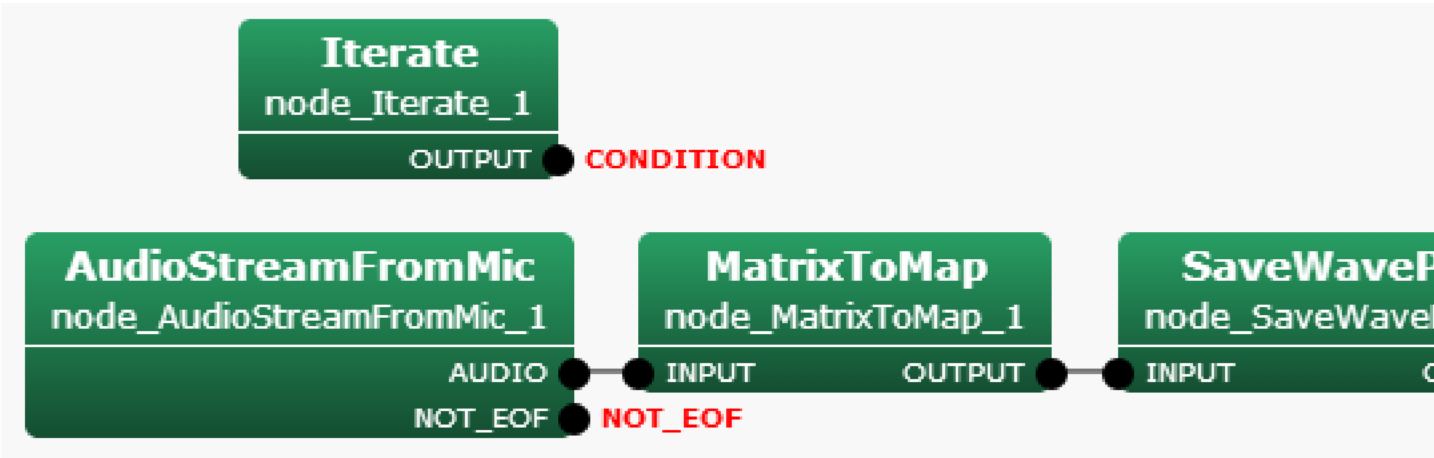


Figure 2.2: MAIN_LOOP (iterator)

Table 2.1: Parameters you need to configure

Node name	Parameter name	Type	Meaning
Iterate	MAX_ITER	`int`	Number of frames to record
AudioStreamFromMic	CHANNEL_COUNT	`int`	Number of channels you use
AudioStreamFromMic	SAMPLING_RATE	`int`	Sampling rate
AudioStreamFromMic	DEVICETYPE	`string`	ALSA
AudioStreamFromMic	DEVICE	`string`	Device name starting with `plughw:`

Running the network :

Now, run the network. You will see multiple files such as `sep_0.wav`, `sep_1.wav`,..., with the number of files being the same as the number of channels. Play the files using audio players such as aplay

```
aplay sep_0.wav
```

Troubleshooting :

If you succeeded in recording, congratulations! If not, check the following and consult the recipe: Sound recording fails.

Microphone connection:
Is the microphone connected properly? Unplug it and plug it in again.

Plug-in power:
Does the microphone need plug-in (external) power? If so, check if the battery has enough energy and if the switch is on.

Use other recording software:
Try to record with another recording software program such as wavesurfer or audacity. If you still fail to record, check the configuration of the OS, the driver, or the device itself.

Device name:
If your computer has multiple sound devices, confirm the name of the device you are using. Check again using `arecord -l`, or try other device names.

Discussion

SaveWavePCM is the easiest way to record a sound. However, you can also use SaveRawPCM if you want a waveform without any headers. SaveRawPCM saves as a raw file whose format is 16 bit little-endian and extension is ".sw" Fig. 2.3 shows the subnetwork using the node.

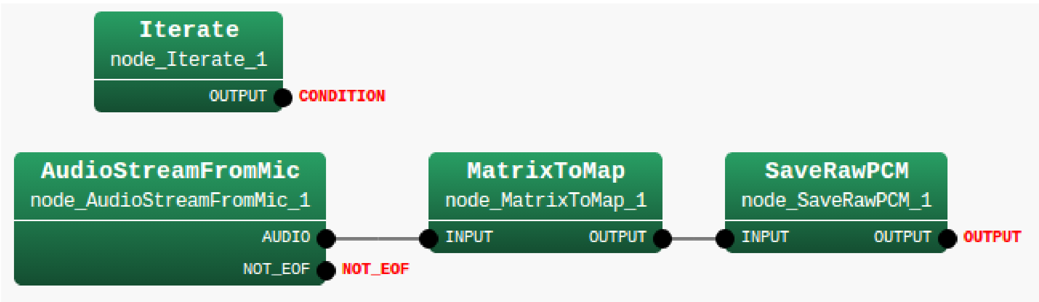


Figure 2.3: MAIN_LOOP (iterator subnetwork)

You can read and visualize the raw file using many programming languages, for example, the python modules numpy and pylab :

```
import numpy, pylab
waveform = numpy.fromfile("sep_0.sw", numpy.int16)
pylab.plot(waveform)
pylab.show()
```

You can play the raw file using `aplay`:

```
aplay -c 1  -r 16000 -f S16_LE sep_0.sw
```

You can convert the raw files (.sw) to wave file using sox. For example, if you want to convert `sep_0.sw` recorded with 16bit, 16kHz sampling, run the following command to get `sep_0.wav`

```
sox -r 16000 -c 1 --bits 16 -s sep_0.sw sep_0.wav
```

See Also

The chapter on devices in the HARK document, as well as the recipe Sound recording fails, are relevant. See the recipe: Recording multichannel sound for sound recording using wios .

See the HARK document for details of nodes and parameters in the network files.

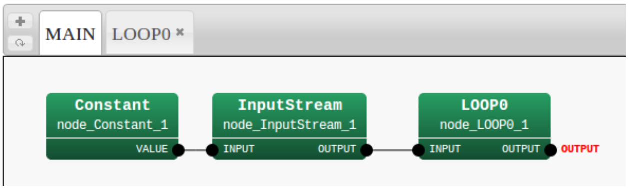
2.2 Learning sound localization

Problem

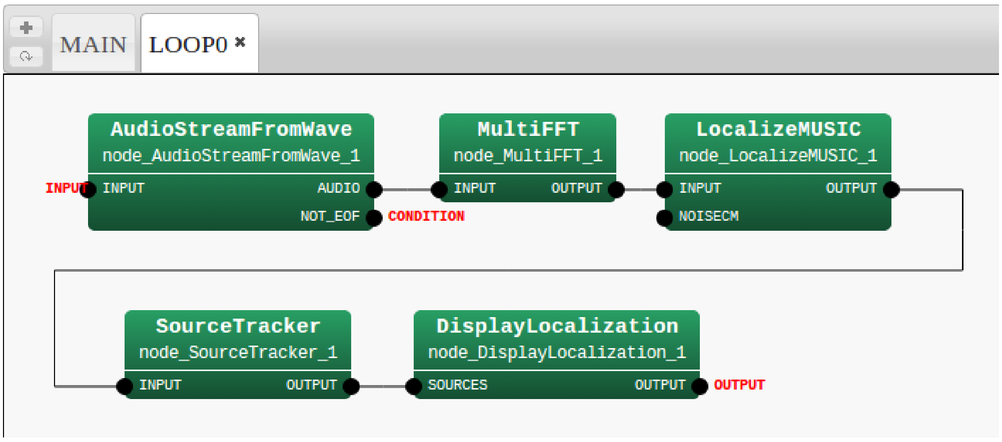
I want to perform source localization in HARK but don't know what to start with.

Solution

(1) Source localization of an audio file



(a) MAIN Subnetwork



(b) Iterator Subnetwork

Figure 2.4: HARK network file for sound source localization using a .wav file

Fig. 2.4 shows an example of a HARK network file for sound source localization using a .wav file input. The .wav file contains multi-channel signals recorded by a microphone array. In the network file, it localizes sound sources and displays their locations.

For the node property settings in the network file, see Section 6.2 in the HARK document.

We provide an example of a HARK network file, called "sep_rec_offline.n", which includes sound source localization at HARK Automatic Speech Recognition Pack.

For the first simple test, download and unzip the HARK Automatic Speech Recognition Pack. Go to the unzipped directory and type the following command.

```
batchflow sep_rec_offline.n 2SPK-jp.wav
```

You will then see a localization result, as in Fig. 2.5. If you see the window and the localization result, the network is working correctly.

(2) Real time sound source localization from a microphone

Fig. 2.6 shows an example of a HARK network file for real-time sound source localization using a microphone array.

Here, AudioStreamFromWave in Fig. 2.4 is replaced by AudioStreamFromMic . By properly setting the parameters in AudioStreamFromMic , a sound source can be localized in real time using a microphone array. For the setting of these parameter, see Section 6.2 in the HARK document. If the network file works properly,

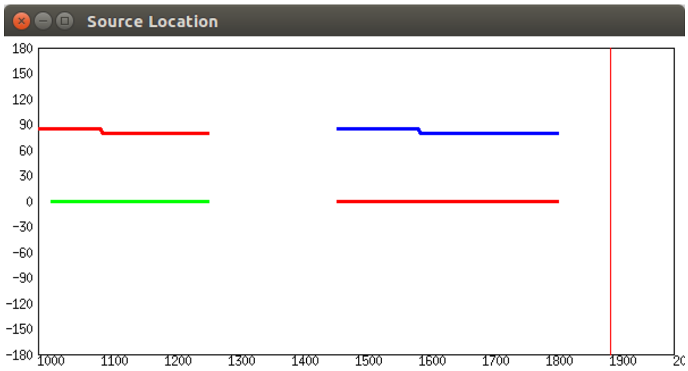
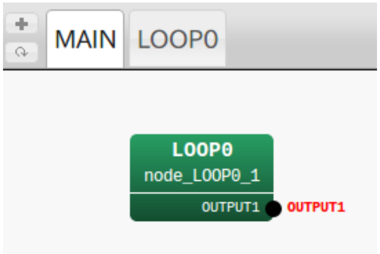
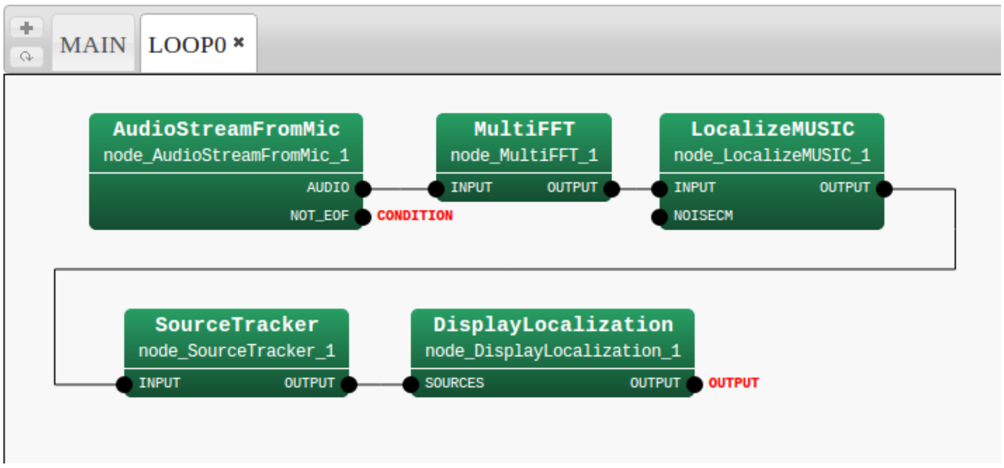


Figure 2.5: Snapshot of the sound source localization result using sep_rec_offline.n



(a) MAIN Subnetwork



(b) Iterator Subnetwork

Figure 2.6: HARK network file for sound source localization using a microphone array

you will see the localization result as inFig. 2.5. If it does not work properly, read , "Sound recording fails" or "Sound source localization fails"

(3) Sound source localization with suppression of constant noise

The sound source localization shown in Fig. 2.4 and Fig. 2.6 can not determine which sound sources are desired. If there are several of high power noise in your environment, LocalizeMUSIC will only localize noise. In the worst case, it cannot localize speech, resulting in a drastic degradation of performance of automatic speech recognition.

This is especially true for automatic speech recognition by a robot-embedded microphone array, in which there are several sources of high power noise related to the robot motor and fan, degrading the performance of the entire system.

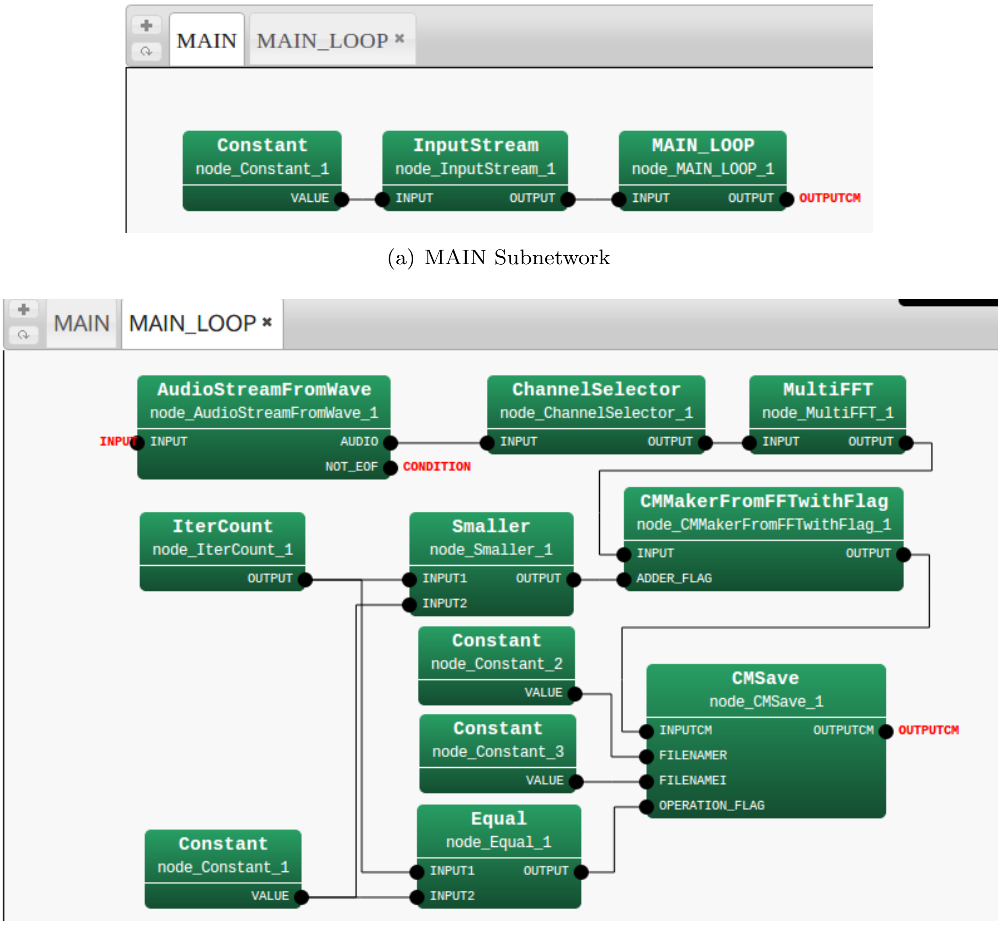
To solve this problem, HARK supports the pre-measured noise suppression function in sound source localization. To enable this function, two steps are needed:

3-1) Generation of pre-measured noise files for localization

3-2) Sound source localization with these noise files

The next two section explain (3-1) and (3-2), respectively.

(3-1) Generation of pre-measured noise files for localization



(a) MAIN Subnetwork

(b) Iterator Subnetwork

Figure 2.7: HARK network file for generating the noise files for sound source localization

Fig. 2.7 shows an example of a HARK network file for generating a pre-measured noise file for sound source localization. To set the parameter of the HARK nodes, see Section 6.2 in the HARK document. The Iterator (LOOP0) subnetwork in Fig. 2.7 has 3 Constant nodes, an IterCount node, a Smaller node, and an Equal node. The parameter settings for those nodes are:

- `node_Constant_1`
  - **VALUE**
    `int` type. VALUE = 200.
    This represents the frame length used to generate the noise file from the first frame.
- `node_Constant_2`

- **VALUE**
  `string` type. VALUE = NOISEr.dat.
  File name for the real part of the noise file.

- `node_Constant_3`
  - **VALUE**
    `string` type. VALUE = NOISEi.dat.
    File name for the imaginary part of the noise file.

- `node_IterCount_1`
  - No parameter
    This outputs the index of the HARK processing frames

- `node_Smaller_1`
  - No parameter
    This determines the index of HARK processing frames is larger than a specific number.

- `node_Equal_1`
  - No parameter
    This determines if the index of HARK processing frames is equal to a specific number.

Here, we set the `node_Constant_1` VALUE at 200. We therefore set `MAX_SUM_COUNT` in CMMakerFromFFTwithFlag as greater than 200.

This network file utilizes a .wav file input containing only noise. Depending on the VALUE of `node_Constant_1`, this node generates noise file for certain frames.

When you run the network file, two files, named NOISEr.dat and NOISEi.dat, will appear in your current directory. These two files are used for sound source localization with noise-suppression function.

In this example, we used 200 frames from the first frame to generate the noise file. By using conditions other than those of the Smaller node, you can specify which frame you will use for the generation.

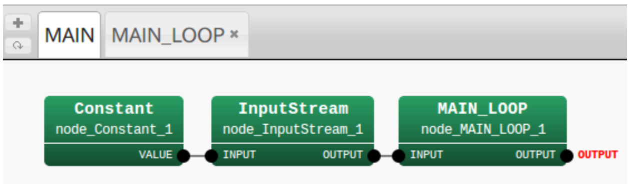
(3-2) Sound source localization with the noise files

Fig. 2.8 shows an example of a HARK network file for sound source localization using noise files created in (3-1), NOISEr.dat and NOISEi.dat. For the parameter settings of the HARK nodes, see Section 6.2 in the HARK document. The Iterator (LOOP0) subnetwork in Fig. 2.8 has 3 Constant nodes, and the parameter setting for those nodes are:

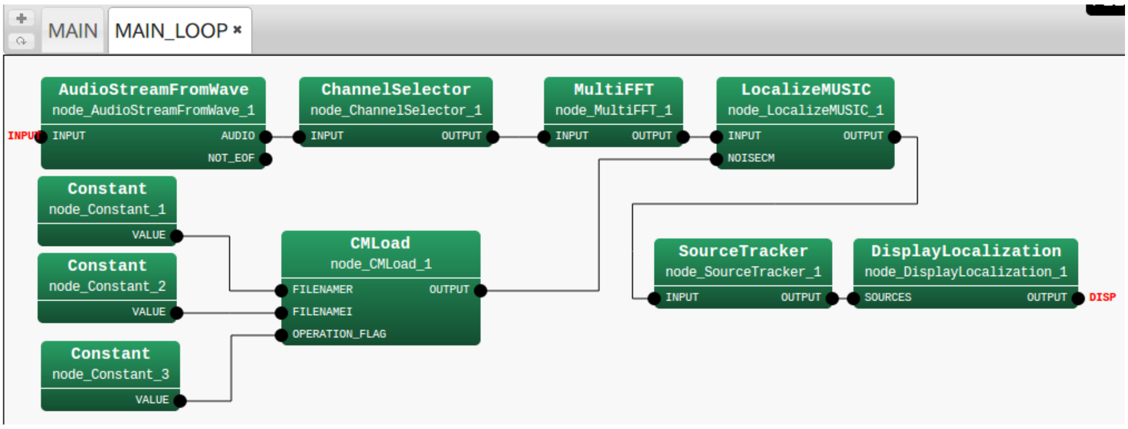
- `node_Constant_1`
  - **VALUE**
    `string` type. VALUE = NOISEr.dat.
    File name for the real part of the loaded noise file.

- `node_Constant_2`
  - **VALUE**
    `string` type. VALUE = NOISEi.dat.
    File name for the imaginary part of the loaded noise file.

- `node_Constant_3`
  - **VALUE**
    `int` type. VALUE = 0.
    This enables updating noise information every frame. If 0, the noise files are loaded only at the first frame.



(a) MAIN Subnetwork



(b) Iterator Subnetwork

Figure 2.8: HARK network file for sound source localization with pre-measured noise suppression

CMLoad reads the noise files, NOISEr.dat and NOISEi.dat, and whitens the noise in sound source localization. To enable the noise suppression function, set `MUSIC_ALGORITHM` in LocalizeMUSIC to GEVD or GSVD. The details of the algorithm for the noise suppression are described in Section 6.2 of the HARK document.

When you run the HARK network file, you will see sound source localization results similar to those in Fig. 2.5. Compared with localization without noise suppression, you will see a greater focus on speech localization.

Discussion

For all the details about the algorithm and noise suppression in LocalizeMUSIC , see Section 6.2 in the HARK document. To increase accuracy, read the recipe in Chapter 8 or the descriptions of the nodes LocalizeMUSIC and SourceTracker in the HARK document and tune it.

See Also

Sound recording fails, Sound source localization fails

2.3 Learning sound separation

Problem

First sound source separation with HARK.

Solution

A network file and either an HGTF binary format transfer function file or a microphone position file are needed to perform sound source separation with HARK.

A network for sound source separation requires four items:

Audio signal acquisition:
AudioStreamFromMic and AudioStreamFromWave nodes can be used.

Source location:
ConstantLocalization , LoadSourceLocation , and LocalizeMUSIC nodes can be used, with ConstantLocalization node being easiest to use. For online processing, use the LocalizeMUSIC node.

Sound source separation:
The GHDSS node can be used for sound source separation. The input consists of source location and the audio signal. The output is the separated signal. The GHDSS node requires a transfer function, which is stored as a HGTF binary format file or calculated from a microphone position file.

Separated signal preservation:
Since a separated signal is in the frequency domain, a user must use a Synthesize node before using a SaveRawPCM or SaveWavePCM node.

HARK supports HRLE-based post-processing.

Post-processing :
Noise suppression can be applied to the separated signal, using the PowerCalcForMap , HRLE , CalcSpecSubGain , EstimateLeak , CalcSpecAddPower , and SpectralGainFilter nodes are used.

Figure 2.9, 2.10, and 2.11 are screenshots of sample networks. Fig.2.10 and Fig. 2.11 show sound source separation without and with post-processing, respectively. These sample networks are extensions of those in Learning sound localization, that is, the added GHDSS node. Either an HGTF or a microphone position file should be specified. To listen to the separated signal, use the Synthesize module before use SaveRawPCM or SaveWavePCM node. When executing these networks, simultaneous speech from two speakers are separated and saved in sep_0.wav, sep_1.wav, ...

Discussion

Offline / Online :
For online separation, replace the AudioStreamFromWave with the AudioStreamFromMic node.

Specific direction / Estimated direction :
To separate sound coming from an estimated direction, use the LocalizeMUSIC node. To separate sound coming from a specific direction, use the ConstantLocalization node. If an estimated sound source location is saved by using the SaveSourceLocation node, it can be loaded using a LoadSourceLocation node.

Measurement-based / Calculation-based transfer fucntion :
To use a measurement-based transfer function, the parameters TF_CONJ and TF_CONJ_FILENAME should be designated as the "DATABASE" and a corresponding file name. To use a calculation-based transfer function, the parameters TF_CONJ and MIC_FILENAME should be designated as "CALC" and a corresponding file name. To use a new microphone array, either a new transfer function or microphone position file is needed

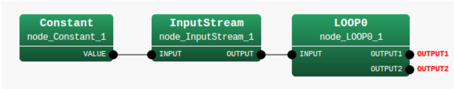


Figure 2.9: MAIN

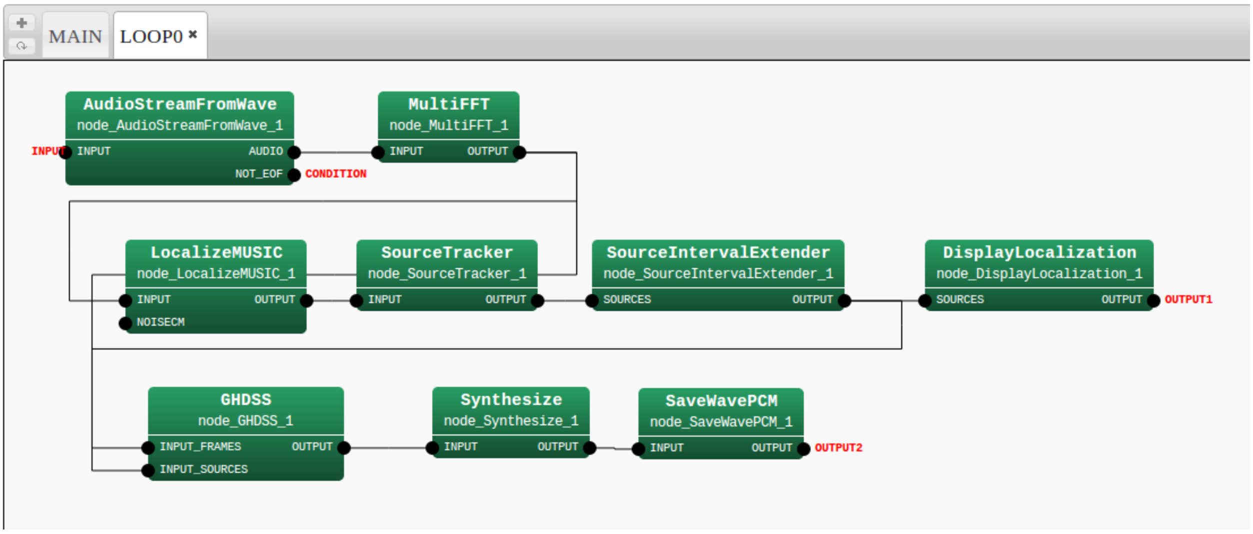


Figure 2.10: MAIN_LOOP (without post-processing)

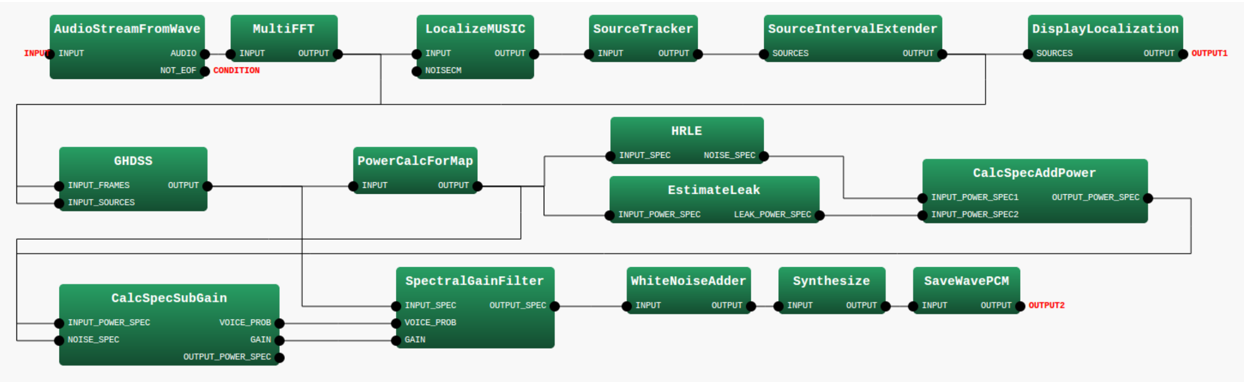


Figure 2.11: MAIN_LOOP (with post-processing)

Online processing may decrease the performance of sound source separation. This can be dealt with by tuning parameters to the the corresponding condition. For parameter tuning, see Sound Source Separation.

See Also

If separation is inadequate, see "Sound source separation fails." Since sound source separation have been 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, the recipes entitled "Learning sound recording", "Learning source localization", "Sound recording fails" and "Sound source localization fails" may be helpful.

2.4 Learning speech recognition

Problem

This is the first time I am trying to recognize speech with HARK.

Solution

Speech recognition with HARK consists of two main processes.

1. Feature extraction from an audio signal with HARK
2. Speech recognition with JuliusMFT

If you are performing speech recognition for the first time, it is better to modify the sample networks of speech recognition, as shown in the Appendix.

Feature extraction :
MSLS and MFCC features are supported by HARK. As an example, we will explain how to extract audio feature consisting of MSLS, Δ MSLS, and Δ power, or MFCC, Δ MFCC, and Δ power. Figure 2.12 and 2.13

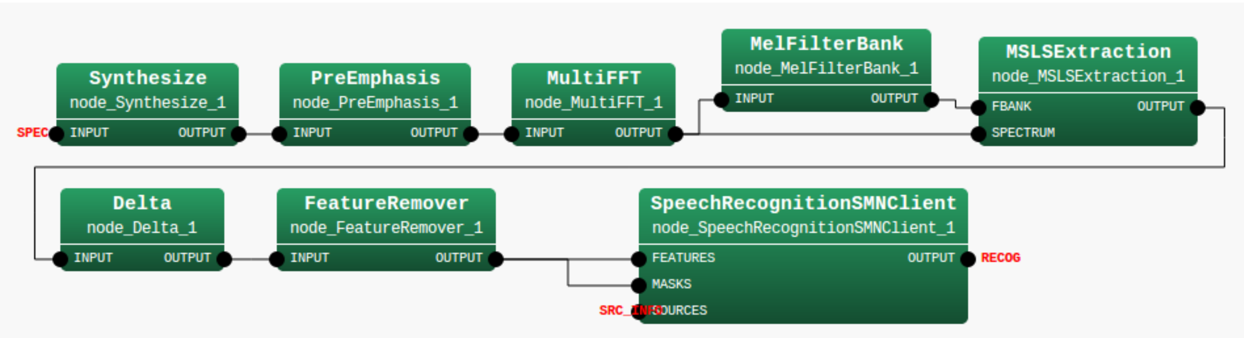


Figure 2.12: MSLS

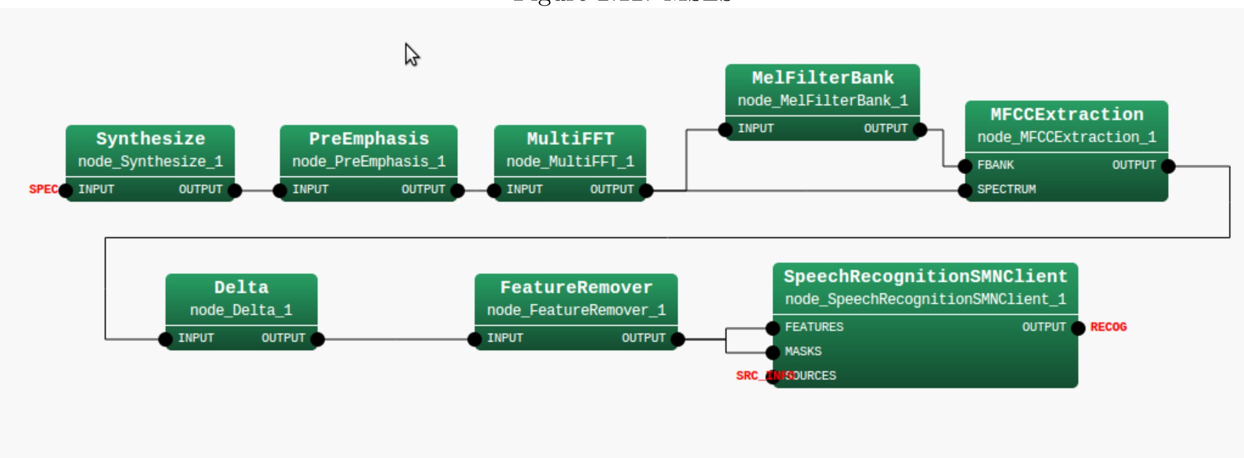


Figure 2.13: MFCC

shows network files to extract MSLS and MFCC features, respectively. PreEmphasis , MelFilterBank , Delta , FeatureRemover and either the MSLSExtraction orMFCCExtraction nodes are used. The SpeechRecognition-Client node sends the extracted feature to JuliusMFT by socket connection. Speech recognition is dependent on sound sources.

To save features, use the SaveFeatures or SaveHTKFeatures node.

Speech Recognition :
JuliusMFT , which is based on Julius, is used to recognize the extracted features. If this is the first time you are using Julius, see the Julius web page and learn the basic usage of Julius.

Use "mfcnet" option for input format when you want to receive features with socket connections from HARK. The following is an example;

```
\begin{verbatim}
-input mfcnet
-plugindir /usr/lib/julius\_plugin
-notypecheck
-h hmmdefs
-hlist triphones
-gram sample
-v sample.dict
\end{verbatim}
```

The first three lines are necessary to receive features from HARK.
Line 1 to receive features from the socket connection,
Line 2 for the plugin enabling the use of the socket connection, Line 3 for MSLS feature.
The "-plugindir" option must be set correctly according to your environment.

Discussion

The simplest method consists of:

- Read monaural sound using AudioStreamFromMic node
- Connect the output of the AudioStreamFromWave node to the input of the PreEmphasis node, as shown in Figure 2.12

If you want to recognize separated sound from the GHDSS node, connect the output of the GHDSS node to the Synthesize node in Figure 2.12 or 2.13.

See Also

Since using JuliusMFT is almost the same as using Julius, the latter manual may be useful. If you want to learn more about the features or models used in JuliusMFT , see Feature extraction or Acoustic and Language Models.

If you perform sound source localization and/or sound source separation, see the recipes entitled Sound recording fails, Sound source localization fails, Sound source separation fails, and Speech recognition fails

Chapter 3

Something is Wrong

3.1 Installation fails

Problem

The standard method of installing HARK is adding the HARK repository using the HARK installation instructions, and running

```
sudo apt-get install harkfd
```

This recipe should be used if the above procedure does not work.

Solution

To install HARK to an unsupported OS, it must be compiled from its source. See the section on Installation from Source Compilation in HARK installation instructions, then download the source, compile, and install.

If installation fails, the remaining HARK files such as its library, will remain in the old version. Uninstall all HARK-related software, then reinstall.

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
```

To check if the old HARK has been properly deleted, execute `flowdesigner` in a command line. If successfully deleted, the error, "The command is not found", will be displayed. Execute `ls /usr/local/lib/flowdesigner` and confirm that the directory has been deleted. If it is still there, delete it. If another directory as been designated for past compilations, delete it if necessary.

Discussion

Major problems with installation may be caused by

1. Software mismatches (Dependency problem)

2. Differences in installation path.

This recipe introduces a method to confirm installation from the beginning. However, it is important to read the error message well before everything else. It is no exaggeration to say that half of all problems can be solved if the user understands the error messages.

See Also

When the installation has been completed successfully, read Learning HARK

3.2 Sound recording fails

Problem

Often a system may not work well because the recording itself has not been performed properly. This section describes

1. A method to confirm that the recording has been performed properly, and
2. The usual reasons that sound recording is not performed properly and measures to correct these problems.

Solution

Try to record after reading the recipe Learning sound recording. It will be easier to confirm later if a user says something for sound recording. Next, confirm that the file has been recorded using software that can display waveforms, such as Audacity and wavsurfer. When recording using more than three microphones, it is convenient to use software that accepts multiple channels. If the display shows that each waveform has been recorded by all channels to which microphones have been connected, the recording has been successfully completed. If there are almost no amplitudes or clipping, confirm that

Each microphone is working properly Connect each to a familiar PC and confirm that each can record sounds.

Each microphone is connected properly Confirm if each is properly connected. Unplug a microphone and plug in again.

The microphone has been plugged in Some types of microphones cannot access a sufficient sound volume without an external power source (called plug in power). Although some sound recording devices supply power, if a microphone requires plug in power and the device does not supply power, an external power supply is required. Confirm by consulting the manuals for both the microphone and the device.

Discussion

Speech recognition systems often do not work properly because the sound itself was not recorded or not recorded properly. particular, for a system with a large number of microphones, it is important to confirm that each microphone can record sounds before operating the software.

See Also

A detailed method of sound recording is described in Learning sound recording. A detailed description of sound recording devices are described in Chapter 7 in the HARK document.

3.3 Sound source localization fails

Problem

My sound source localization system does not work well.

Solution

This recipe introduces the usual debugging procedures for sound source localization systems.

Confirm the recording

Make sure the system can record sound. See Sound recording fails.

Offline localization

Try offline localization using recorded wave files. Walk around the microphone array while speaking and record the sound. This is your test file.

Replace the node AudioStreamFromMic with AudioStreamFromWave in your network for offline localization. When DisplayLocalization and SourceTracker are connected, the localization result can be visualized.

If this step fails, set the parameter DEBUG to ON of LocalizeMUSIC and check the MUSIC spectrum. This value should be larger in the presence than in the absence of sound. If not, the transfer function file may be wrong. If the spectrum seems successful, adjust the parameter THRESH of SourceTracker so that its value is bigger than in the non-sound area and smaller than in the sound area.

You may want to adjust other parameters;

1. Set NUM_SOURCE for the condition that the system is intended for.
   For example, if there are up to two speakers, set NUM_SOURCE to 2.
2. Set MIN_DEG and MAX_DEG.
   If localization results are obtained from the direction where the sound is not located, there may be reflection from the wall or a noise source (e.g. fans of a PC). If you can assume that the target signals do not come form these directions, the localization result can be set such that they are not output from them using the parameters MIN_DEG and MAX_DEG.

   For example, if no sound will come from behind the robot, then the localization target can be set to only the front side by setting MIN_DEG to -90 and MAX_DEG to 90.

Online localization

To check the entire system, use the original network file having AudioStreamFromMic and run your program. You may need to tune the THRESH of SourceTracker because of the different position of the talker or volume of the talker's voice. If the system is too sensitive, decrease the THRESH, otherwise, increase it. The amount might be around 0.1 - 0.2.

Discussion

It may be appropriate to change parameters, depending on reverberations in the room and the volume of the speaker's voice. To increase the localization performance of the current system, it may be better to tune up in the actual location. Since THRESH is the most important parameter here, adjusting it only will be the first thing to try.

See Also

If you are new to building a sound source localization system, refer to Learning sound localization. The chapter Sound source localization includes some recipes for localization. If you want to know the detailed algorithm, see LocalizeMUSIC and SourceTracker in the HARK document.

3.4 Sound source separation fails

Problem

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

Solution

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

Confirm that the source localization has been successfully completed Since the GHDSS module uses source localization results (number of sound sources) for inputs, the separation performance degrades if localization failed. Modify the network files so that the localization results are displayed and confirm whether the system localizes sounds successfully. For further information, see the recipe, Sound source localization fails.

Confirm that GHDSS separates sounds properly To determine whether the GHDSS has been performed properly, save a result of GHDSS using SaveWavePCM and confirm that separation was successful. If separation failed in GHDSS , see Parameter tuning of sound source separation

Confirm that post-filtering was performed properly When PostFilter or HRLE are inserted just after separation processing, separation may fail due to improper parameters. Save the prost-filtered sound and confirm that the post-filtering was successful. If the post-filtering failed, see Reducing the leak noise by post processing

Discussion

None.

See Also

Sound source localization fails
Parameter tuning of sound source separation
Reducing the leak noise by post processing

3.5 Speech recognition fails

Problem

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

Solution

If you have not yet tried the recipes in Chapter 2, starting with this chapter may be easier for you. In this chapter, the user will learn about a speech recognition system with HARK, in the order of sound recording, sound source localization, sound source separation and speech recognition.

Next, an inspection method will be described if an original system is developed by the user does not work. 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 that HARK and FlowDesigner were installed properly (Recipe Installation fails), that sound is recording properly (Recipe Sound recording fails), that sound source localization is working properly (Recipe Sound source localization fails), and that sound source separation is working properly (Recipe Sound source separation fails) in each recipe.

If you verify that the system works properly through this stage, then, speech recognition must be verified. We presume that Julius (http://julius.sourceforge.jp/), a large vocabulary continuous speech recognition engine, which has been modified for HARK, is used in this system. Three files are important for using Julius.

1. Acoustic model: A model indicating the relationships between features and phonemes of acoustic signals
2. Language model: A model of the language spoken by the user of the system.
3. Configuration file: File names of these two models

If the system does not work at all or Julius does not start, a wrong path may have been designated for the file. Thus, the configuration file should be checked, for details, see Recipe Making a Julius configuration file(jconf). To verify the acoustic model, see Creating an acoustic model; to verify the language model, see Creating a language model.

Discussion

This solution is for the system which does not recognize sounds at all. If the system works properly but its success (=recognition) rate is low, the system must be tuned up. Tuning up a speech recognition system involves complicated problems of localization, separation, and recognition, among others.

For example, see parameter tuning of GHDSS (Recipe Parameter tuning of sound source separation) for separation and tuning of PostFilter (Recipe Reduce the leak noise by post processing). Also see the chapter 10, which discusses features used for recognition, and recipes for improving performance (Tuning parameters of sound source localization and Parameter tuning of sound source separation).

See Also

Recipes only for when the system shows no recognition are shown here

1. Each recipe in the Chapter Something is wrong
2. Acoustic model and Language model.
3. Making a Julius configuration file (.jconf)
4. Julius Book

3.6 Making a debug node

Problem

How to create a module to debug the system I created.

Solution

Basically, printing a standard output is a way to debug your own system.

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

Furthermore, when executing a created network file (called `nfile.n` here), use this command line, which is not from GUI of FlowDesigner ,

- `./nfile.n > log.txt`
  The message (above `message1`) output by `cout` is saved in `log.txt`.
- `./nfile.n 2> log.txt`
  The message (above `message2`) output by `cerr` is saved in `log.txt`.

Some nodes have a parameter `DEBUG` such as LocalizeMUSIC . Messages are output only when `DEBUG` is set a `true`.

3.7 Using debug tools

Problem

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

Solution

Two principal methods are available.

1. Debug using gdb in a command line
2. Error checks of spatial transfer functions using the harktool

The harktool is designed to improve localization and separation performance rather than for debugging the program. Brief descriptions are given for each of these method.

1. **Debug using gdb in a command line**
   gdb is a free source level debugger of GNU and is useful for stopping executions by designating a break point or perform executions for each line. It is used as follows for debugging operations in HARK.
   - Compile with `-g -ggdb` options
     (Add `AM_CXXFLAGS = -g -ggdb` to `Makefile.am`)
   - `gdb /path/flowdesigner`
     Move to the gdb console.
     `(gdb)`
   - Setting of break pointers
     `(gdb) b MODULE::calculate` (when designating by function name)
     `(gdb) b module.cc: 62` (when designating by line count;e.g., the 62nd line of `module.c`)
   - Confirmation of the break points
     `(gdb) i b`
   - Break points with conditions (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 checks of spatial transfer functions using the harktool**
   For (2), see the description in the section of harktool in this manual.

See Also

To create a debug module with gdb, see Making a debug node and "Implementation: Modification of Channel Selector" of the HARK training session material

3.8 Checking a microphone connection

Problem

The system does not work well. The microphone connections may be flawed.

Solution

To confirm that the microphones are connected, it is necessary to examine both the microphone and sound recorder. Connect the microphone to another PC; if recording is successful, the sound recorder of the first PC is the cause of your system failure. Connect another microphone to the sound recorder; if recording is successful, the microphone is the cause of your system failure. Check each microphone and recorder stepwise. For details, see Sound recording fails

Discussion

When recording is unsuccessful, it is important to identify the points of the cause. Look for it patiently.

See Also

Sound recording fails

Chapter 4

Microphone Array

4.1 Selecting the number of microphones

Problem

Read this section when mounting microphones on your robot.

Solution

Theoretically, arbitrary sound sources can be separated by 1 more microphone than the number of sound sources. In actuality, the performance of a system will depend on the layout and operating environment of individual microphones. It is rare to know the number of sound sources a priori, so the optimal number of microphones must be determined by trial and error. It is therefore necessary to select the minimum number of microphones required for sound source localization, sound source separation, separated sound recognition. Theoretically, the optimal layout for recording from sound sources in all directions is an equally-spaced layout on the arc of a concentric circle. We presume here that the head of a robot is a perfect sphere; if not, it might result in less than optimal sound source localization and separation in the direction at which the head shape shows a discontinuous reflection. It is better to set a concentric circle in the head, at which continuous reflection occurs.

Discussion

In our three-speaker simultaneous utterance recognition demonstration, three sound sources are separated with eight microphones. The number of separable sound sources is thus below half the theoretical value, 7. The number of microphones may be increased to, for example, 16 to improve performance and separate a larger number of sound sources. However, over-close microphone intervals between microphones have little effect on performance, while increasing calculation costs.

See Also

None.

4.2 Selecting the layout of the microphone array

Problem

Read this section when mounting microphones on your robot.

Solution

The precondition for layout is that the relative positions between individual microphones do not vary. The appropriate layout for accuracy of localization and separation depends on the directions of sound sources. If a sound source is located in a specific, known direction, the microphones can be positioned close to that direction. It is better to place microphones at wide intervals and perpendicular to the normal vector of the specific direction. If sound sources are located in all directions, then the microphones should be positioned in a circular pattern. Since wider inter-microphone intervals are better for sound source separation, the microphones should be dispersed as far apart as possible. A layout of microphones poor in sound source localization can result in poorer sound source separation. Therefore, the layout should be dependent on sound source localization. If localization accuracy is not good, reconfigure the layout so that it is not in a shape in which acoustic reflection becomes discontinuous around microphone positions, and avoid such locations for positioning.

Discussion

When the relative position between microphones varies, so will the impulse response of a microphone array. We presume here that the impulse responses of a microphone array are fixed in HARK.

See Also

None.

4.3 Selecting types of microphones

Problem

Read this section if you are unsure what type of microphone to use.

Solution

We have used non-directional electric condenser microphones, which cost only several thousand yen each. It is not always necessary to use more expensive microphones. Microphones with a higher signal-to-noise ratio are preferable. It is better to choose wires with proper covering.

Discussion

To record sounds with a sufficiently high signal-to-noise ratio, choose wires with proper covering and with plugs. Although this type of microphone is usually expensive, we have confirmed that this system works with microphones costing several thousand yen each. When wiring inside a robot, make sure that all wires are covered and that interference by other signal wires is suppressed, rather than using expensive microphones.

See Also

None.

4.4 Installing a microphone array in a robot

Problem

Read this section when installing microphones in your robot.

Solution

To wire microphones, shorten the lengths of their wires. In installing wiring inside the robot, make sure that these wires are not 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 the microphones so that they are in contact with the surface of the robot's housing. That is, make sure the microphones are embedded in the housing, with only the tips are outside the robot.

Discussion

When microphones are isolated from the surface, they are affected by reflections from the robot housing, leading to the degradation of the performance of sound source localization and sound source separation. If the housing vibrates during robot operations, make sure the microphones do not pick up these vibrations. For mounting the microphones, use materials that suppress vibrations, such as bushings.

See Also

None.

4.5 Selecting a sampling rate

Problem

Read this section if you do not know how to determine a sampling rate in entering acoustic signals from a device.

Solution

In the absence of particular indications, 16kHz is a sufficient sampling rate. If there are particular indications, use the lowest sampling rate within the range at which alias does not occur. The frequency that is a half the sampling rate is called the Nyquist frequency. Alias occurs in signals with frequency components greater than the Nyquist frequency. Therefore, to prevent alias in input signals, it is preferable to use as high a sampling rate as possible. At the highest sampling rate, alias is infrequent. When the sampling rate is increased, however, the amount of data to be processed increases, increasing calculation costs. Therefore, when setting a sampling rate, it is better not to increase it by more than the amount necessary.

Discussion

Speech energy reaches over 10kHz in bandwidth, with much of the energy present as low frequency components. Therefore, consideration of low frequency bands is often sufficient for most purposes. For example, for telephones, sampling ranges from around 500 Hz to 3,500 Hz , making possible the transmission of interpretable audio signals with bandwidths of up to around 5kHz [1] . For 16kHz sampling, frequency components $\leq$ 8kHz can be sampled without alias, making it useful 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.

4.6 Using an A/D converter unsupported by HARK

Problem

Read this section if you wish to capture acoustic signals using devices other than the sound cards supported by ALSA (Advanced Linux Sound Architecture), the RASP series (System In Frontier, Inc.), and TD-BD-16ADUSB (Tokyo Electron Device), all of which are supported by HARK as standards.

Solution

An A/D converter can be used by a corresponding node created by the user. We describe here a procedure for creating an AudioStreamFromMic node that supports, for example, NewDevice. The overall procedure consists of:

1. Creation of a class NewDeviceRecorder corresponding to the device and placing its source and header files into the librecorder directory in the HARK directory.
2. Rewriting AudioStreamFromMic so that the created class can be used.
3. Rewriting Makefile.am, configure.in to compile.

In (a), a class is created that takes data from the device and sends them to a buffer. This class is regarded as the successor to the Recorder class. Initialization prepares the device for use, followed by the operator() method, which removes data from the device. In (b), processing is performed when "NewDevice" is designated as the option is described. The NewDevice is designated and initialized in the constructor, and the signals are set. In (c), Makefile.am and configure.in are changed to correspond to the newly added file. Described below is a sample of AudioStreamFromMic 2, 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 that was spun off from the Recorder class, enabling the NewDevice to be used. We describe the processing required to connect with the device with an initialize function and to read data from the device to the () operator. This source code (NewDeviceRecorder.cpp) is created 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 AudioStreamFromMic module performs only the processing required to read the contents of the buffer belonging to the Recorder class. Data exchange with devices are performed by each class (ALSARecorder, ASIORecorder, and 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 constructing a node are described in Section 12.1, "How to create nodes?".

Chapter 5

Input Data Generation

5.1 Recording multichannel sound

Problem

To record multichannel sound from a microphone array.

Solution

HARK-supported devices

Multichannel recording requires an audio device supporting multichannel input. HARK currently supports the following devices (for details, see the chapter on devices in the HARK document).

ALSA Audio device that can access via Advanced Linux Sound Architecture (ALSA),

RASP RASP series, System In Frontier, Inc.

TDBD TD-BD-16ADUSB, Tokyo Electron Device Ltd.

Two recording methods

Using HARK, sound can be recorded in two ways , by using a HARK network, and by recording using wios , a support tool provided by HARK.

Recording using a HARK network

HARK provides two nodes AudioStreamFromMic , which gets sound from a recording device, and SaveRawPCM / SaveWavePCM , which saves waveforms. You can build a recording network by connecting these nodes directly (see Learning sound recording for details about networks, and HARK documents for details about these nodes).

Recording using wios

HARK provides a support tool wios for recording/playing sounds through ALSA, RASP, and TDBD. The ability of wios to synchronously play and record a sound enables its use for measuring impulse responses, i.e., propagation from a sound source to a microphone, because it can to play a special signal and record the sound simultaneously. This recipe describes how to use wios for recording; to use wios to measure impulse responses, see Recording impulse response. You can install wios with the following command if you registered the HARK repository in your system. (See HARK installation instructions).

```
sudo apt-get install wios
```

wios options has four categories; for a detailed description, see wios help by running wios with no alterations.

Mode options :
Three modes: playing mode(`-p`), recording mode(`-r`), and synchronized-playing-and-recording mode(`-s`). Use `-t` to specify the duration.

File options :
File name for playing, (D/A) (`-i`); for recording (A/D) (`-o`).
Quantization bit rate (`-e`), sampling frequency(`-f`), number of channels(`-c`)

Device options :
Device type specification (`-x`) (ALSA:`0`, TDBD:`1`, RASP:`2`)
Device specification (`-d`). The meaning of this option depends on which type of device is specified. `-d` is the device name for ALSA (default: `plughw:0,0`), TDBD (default: `/dev/sinchusb0`), and the IP address for RASP (default: 192.168.33.24).

Device-dependent options :
Examples: ALSA: Buffer size; TDBD: Gain; RASP: Gain. See `wios` help for a complete list.

Examples:

- Using a RASP device with an IP address of 192.168.1.1, ...
  1. Record sound with 8 channel for 3 seconds and save it to `output.wav`.
     `wios -r -x 2 -t 3 -c 8 -d 192.168.1.1 -o output.wav`
  2. Play the wave file `input.wav`
     `wios -p -x 2 -d 192.168.1.1 -i input.wav`
  3. Play `tsp.wav` and synchronously record the sound and save it to `response.wav`
     `wios -s -x 2 -d 192.168.1.1 -i tsp.wav -o response.wav`
- Using an ALSA-supported sound card installed in a computer, ...
  1. Record monaural sound for 10 seconds
     `wios -r -x 0 -t 10 -c 1 -o output.wav`
  2. Play `output.wav`
     `wios -p -x 0 -i output.wav`

See Also

See Learning sound recording for a detailed description of sound recording sound using the HARK network. If recording fails, see Sound recording fails for troubleshooting.

For ALSA, RASP, and TDBD, see the chapter about devices in the HARK document. If you want to use a device unsupported by HARK, see Using an A/D converter unsupported by HARK.

See Recording impulse response for information about impulse response measurement.

5.2 Recording impulse response

Problem

To measure impulse responses for sound source localization and sound source separation.

Solution

Ingredients

You need a loudspeaker, a microphone array, an audio device that can simultaneously play and record a sound, and a signal including multiple TSPs. Upon installing harktool , a wave file of a single TSP signal is installed in `/usr/bin/harktool3_utils`; the file name is `16384.little_endian.wav`. Since this file includes a single TSP signal, it is necessary to cut and paste to make multiple TSPs (e.g., 8 or 16 times) using waveform editing software, such as matlab, or python. Note that NO spaces should be inserted between TSPs.

You can decide the number of TSPs depending on the environment of the measurement. If the environment is quiet, you can use 8-times TSPs which is short. However, if the environment is noisy, e.g., an air conditioner is running, we recommend to use 16-times TSPs to suppress the effect of the noise.

Similarly, `16384.little_endian.tsp` is also installed to the same path. This is a raw file, with32 bit floating amplitude data installed in series. To read the file, use `wavesurfer` (to install on Ubuntu, use `sudo apt-get install wavesurfer`) Then, open the file with `wavesurfer`,

```
wavesurfer /usr/bin/harktool3_utils/16384.little_endian.tsp
```

and set the configuration (Fig. 5.1)

Directions

Impulse responses are measured in two steps: (1) playing and recording the TSP signal, and (2) calculating impulse responses from TSP signals.

1. Playing and recording the TSP signal
   Play multiple TSP signals because their addition reduces background noise and reverberation. Assume the use of `tsp8.wav` having eight TSP signals.

   Use wios for recording. If you want to record the impulse response using an ALSA device with 8 channels, use the following command (see Recording multichannel sound how to use wios)

   ```
   wios -s -x 0 -i tsp.wav -c 8 -o r100_d000.wav
   ```

   It is necessary to record impulse responses from every direction. For example, impulse responses can be measured along a circle with its center at the microphone array position and a radius of 1-2m. After each recording, the loudspeaker should be moved 5 or 10 degrees, followed by another recording. If impulse responses are recorded at a 1m distance and every 10 degrees, the file names will be

   ```
   r100_d000.wav
   r100_d010.wav
   ...
   r100_d350.wav
   ```

2. Calculating impulse responses from TSP signals.
   The impulse response for each TSP signal can be calculated by adding each TSP and calculating the converse of the inverse TSP signal with the added TSP. These calculations can be performed using harktool. See the HARK document or the tutorial video on measurements of transfer function for details.

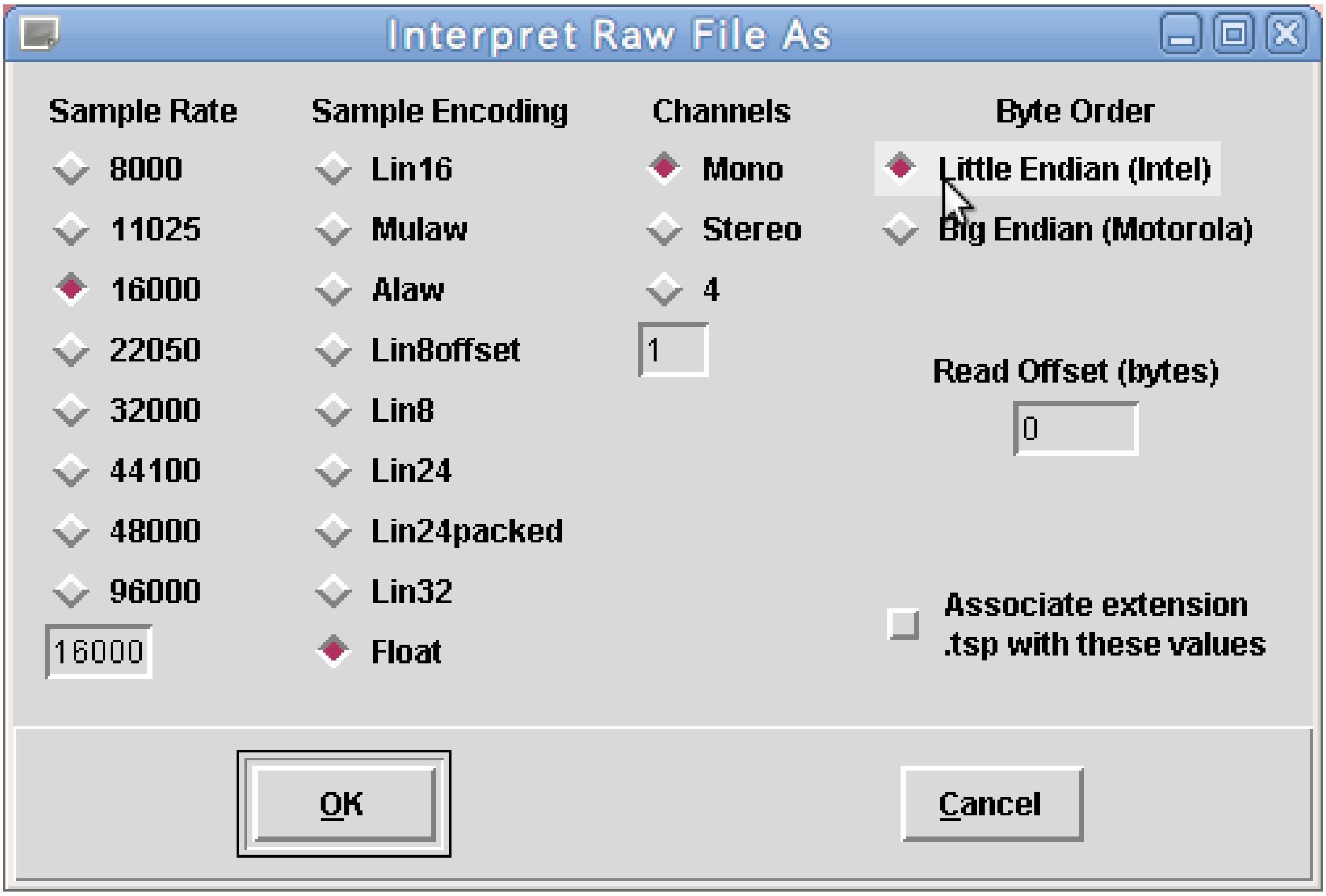


Figure 5.1: Configuration of wavesurfer needed to read raw files

Discussion

Impulse response is an output of the system following the application of an impulse function to the system. Intuitively, the reverberation you hear upon clapping your hands in a room is a type of impulse response. The problem in actually measuring the "impulse response" is the need for a large amount of energy with a sufficient signal-to-noise ratio. To solve this problem, the impulse response is measured using a Time Stretched Pulse (TSP) signal, whose energy is stretched over time. Impulse response can be determined by recording the TSP and calculating the converse and inverse of the TSP [1,2]. The best interval between measurements is usually around 5 or 10 degrees, although it depends on the microphone array and on the shape and configuration of the room.

See Also

See Recording multichannel sound to determine how to use wios . See HARK Installation Instructions to determine and the section on harktool in the HARK document on the installation of HARK and the use of the harktool. We also provide an instruction video on the measurement of transfer function .

References

(1) Y. Suzuki, F. Asano, H.-Y. Kim, and Toshio Sone, "An optimum computer-generated pulse signal suitable for the measurement of very long impulse responses", J. Acoust. Soc. Am. Vol.97(2), pp.-1119-1123, 1995

(2) Impulse response measurement using TSP (in Japanese) http://tosa.mri.co.jp/sounddb/tsp/index.htm

5.3 Synthesizing multichannel sound from impulse response

Problem

To synthesize multichannel sound to test my system offline

Solution

You can synthesize a multichannel sound with an original sound and an impulse response file. Multichannel sound is synthesized by the convolution of an original sound and an impulse response. Using FFT-based implementation such as cyclic convolution, you can obtain the simulated data quickly. The pseudo Matlab code for convolution is:

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

Here, we assume that SampleData1ch.wav is the 1 channel original speech in Microsoft RIFF format, with imp1, ..., imp4 indicating time-domain representations of impulse responses from the sound source to the microphone 1, ..., 4. Thus, y1,y2,y3 and y4 are the multi-channel synthesized sound being simulated. It is important to confirm impulse responses and sampling frequency of the original sound before synthesizing because the convolution of sounds with different sampling rates is meaningless. To generate a mixture of sounds, add the synthesized sounds.

Discussion

The convolution of an original sound and an impulse response can simulate multiplicative noise. This multiplicative noise, such as transfer characteristics from the sound source to the microphone or the recording device, is modeled by the convolution to the clean signal. Therefore, this synthesis simulates multiplicative noise.

See Also

To measure impulse response, see Recording impulse response. To add noise, see Adding noise.

5.4 Adding noise

Problem

To add noise from a robot fan or motor for simulation.

Solution

To apply the robot audition system to the real world, noise from the surrounding environment, the system itself, and/or a recording device must be considered. By recording these noises in advance and adding them to the original sounds, you can evaluate your system more realistically. Adding noise is simple; First, synthesize multichannel sounds using the recipe: Synthesizing multichannel sounds from impulse responses. Then, add the recorded noise. Matlab pseudo codes are shown below.

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

where signal.wav denotes the original sound, noise.wav denotes the noise, and z is the simulated sound with additive noise.

Discussion

Additive noise is noise that is added to the original sound, for example robot fan noise.

See Also

To simulate multiplicative noise, see Synthesizing multichannel sound from the impulse response.

Chapter 6

Acoustic and language models

6.1 Creating an acoustic model

Problem

To describe a method of constructing an acoustic model for speech recognition. This type of model can improve speech recognition performance after having the introduction of HARK into a robot.

Solution

An acoustic model is a statistical expression of the relationship between a phoneme and acoustic features and can have a substantial impact on speech recognition. A Hidden Markov Model (HMM) is frequently used. When changing the microphone layout on a robot or the algorithm and parameters for separation and speech enhancement, the properties of the acoustic features input into speech recognition may also change. Therefore, speech recognition may be improved by adapting an acoustic model to new conditions or by creating a new acoustic model that meets these conditions. We describe here the methods of construction of three acoustic models:

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

In each model, HMM is described with a Hidden Markov Model ToolKit (HTK), which is used when creating acoustic models of the speech recognition engine Julius used in HARK.

Although acoustic models have various parameters, we describe here the training of triphone HMM for three states and 16 mixtures. For more information about each parameter, consult textbooks such as the "HTK Book" and the "IT Text speech recognition system". A fundamental flow of the creation of a typical triphone-based acoustic model is shown below.

1. Extraction of acoustic features
2. Training of a monophone model
3. Training of a non-context-dependent triphone model
4. Status clustering
5. Training of a context-dependent triphone model

Acoustic features extraction

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

```
% 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.

1. **Data revision:** Generally, even when using a distributed corpus, it is difficult to completely remove fluctuations in description and descriptive errors. Although these are difficult to notice beforehand, they should be revised as soon as they are found since such errors can degrade performance.

2. **Creation of words.mlf:** Create words.mlf has file names for (virtual) labels that correspond to features and utterances included in files written per word. The first line of each words.mlf file must be #!MLF!#. After describing the labeled file names with " " on the second line, the utterance included in the labeled file name is divided into each word, with the words described on individual lines.

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

   -exec /phonem/rom2mlf
   ; > words.mlf

3. **Creation of word dictionary:** A word dictionary is created that relates words with phoneme lines. Generally, a dictionary with registered word classes is often used. For a small dictionary, it would be sufficient 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):** Phoneme MLFs are created with a dictionary and word MLF. Use LLEd concretely. Rules are described in phones1.led. The rule allowing sp (short pose) is described in HTKBook.

   ```
   % 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 phonemes from words. 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, with one file name per 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 include only normal files.

Preparation of triphone

Although this operation may be performed after training of monophones, it may be necessary to remake phones1.mlf depending on the results of the checking. To save time, this operation can be performed here.

1. Creation of **tri.mlf:** First, create phonemes in triplicate.

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

   Remove the phonemes described in mktri.led from the phoneme context.

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

   Parameters are reduced with short vowel contexts by identifying the anteroposterior long vowel contexts. An example of a created tri.mlf is shown here.

   ```
   ------------------------------
   #!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 triplicates 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 the phoneme contexts but 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 a prototype (proto) of HMM:** The proto can be created in HTK with the tool MakeProtoHMMSet. % ./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
```

These steps result in the creation under hmm0/ of a proto and vFloor (initial model) that learned dispersion and means from all the training data. Note the time required to complete this operation, although it is dependent on the volume of data.

1. **Creation of initial monophones:**

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

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

The monophone1.list is a list of phonemes including sp. In the HTKBook, the "monophone1.list" should 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 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, add the following as a header of macro. Generally, the description of the header should be the same as that of hmmdefs; i.e., dependent on the content of proto.

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

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

Perform repeated training a minimum of three times. (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
```

Although alignment settings should be readjusted at this point, it has been 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 repeated training around 10 times.

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 can be described as follows. Stats on the first line is an output file obtained in 9.2. Replace these with a value around 1000 temporarily 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 with the same status are included.

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. **Increasing the number of mixtures (example of 1 $\rightarrow$ 2mixtures):**

```
% 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 increasing 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, we recommend that the number of mixtures be doubled. ($2 \rightarrow 4 \rightarrow 8 \rightarrow 16$)

See Also

HTK Speech Recognition Toolkit. For acoustic models for Julius, see source information of this document. For acoustic model construction for HTK, see Acoustic model construction for HTK For acoustic model construction for Sphinx, see acoustic model construction tutorial for Sphinx developed by CMU, acoustic model construction for Sphinx.

6.2 Creating a language model

Problem

To describe a method for constructing language models for speech recognition.

Solution

Creation of a dictionary for Julius

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

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

Prepare a right text file and call it seikai.txt. Since it is used for language model creation, insert <s>, </s> at the beginning and end of each sentence, 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 the contents of text.keitaiso; if any part of the morphological analysis is incorrect, revise it. Moreover, since the notion and reading of "he" and "ha" differ, alter their reading to "e" and "wa", respectively. It may be necessary to normalize of morphemes and remove other unwanted parts. These steps are 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 kana2phone_rule.ipa - vocab2htkdic

Creation of language model for Julius

For creation of a language model, see “Speech recognition system” (Ohm sha). To create 2-gram and reversed 3-gram such as jconf of samples, however, use of the CMU-Cambridge Toolkit alone is not sufficient, requiring the use of palmkit, which is compatible with CMU-Cambridge Toolkit. Moreover, the reversed 3-gram has become unnecessary for Julius; therefore, it may be not always necessary to use palmkit. To use palmkit, prepare a correct answer text, designating it seikai-k.txt. This file requires morphological analysis; i.e., punctuation is regarded as a word, with words separated by spaces. <s> and </s> are inserted at the beginning and end of each sentence, respectively, remove transition over <s> and </s> . In this case, descriptions of <s> and </s> are 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
```

The 2-gram and reversed 3-gram are created and all are collected. 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 7

FlowDesigner

7.1 Running the network from the command line

Problem

Although FlowDesigner can be built and executed, it is annoying to use GUI every time. I want to run a network file using a command line, as well as to change network parameters using command line arguments.

Solution

You have to change a parameter value that you want to specify as a command line argument so that the program can associate with the parameter and command line argument.

Open the network file using FlowDesigner , and edit it as follows:

1. In the MAIN subnetwork, open the node property window to which you want to give a command line argument.
2. Designate the Type of variable for which an argument is substituted for in `subnet_param` and enter "ARG?" as Value (the index of the argument should be entered in ?. If this is the first argument, enter "ARG1"). If you want to give the argument as `int` or `float` , you have to specify it (i.e. "int:ARG1" or "float:ARG1").

Note that you can give the command line argument only to the nodes in the MAIN subnetwork. Two steps are required to give command line arguments to other nodes: First, set the Type of the parameter as `subnet_param` so that you can see the parameter in the MAIN network. Then, set the command line argument in the MAIN subnetwork. Then, you can run it as if it were an ordinal executable.

`batchflow` executes the contents of the network file. Therefore, you have to type "batchflow" before the network file and a command line argument.

```
$ batchflow foo.n 0.5 0.9
```

Then, ARG1 is set to 0.5 and ARG2 to 0.9, followed by the execution of `foo.n`.

For Ubuntu

You can also execute the network file as follows:

```
$ ./foo.n 0.5 0.9
```

Then, ARG1 is set to 0.5 and ARG2 to 0.9, followed by the execution of `foo.n`.

For Windows

You can also execute the network file as follow:

```
$ qtflow foo.n 0.5 0.9
```

Then, ARG1 is set to 0.5 and ARG2 to 0.9, followed by the execution of `foo.n`.

Discussion

FlowDesigner is a GUI for editing a network file and the actual processing is performed by `batchflow` in Ubuntu and `qrflow` in Windows (when you type "batchflow," qtflow is acutally performed). Therefore, you can execute a network file by the same way in both Ubuntu and Windows.

The arguments are interpreted as `string` type by default. If utilize different types, such as `int` and `float` types, designate them as "`int` :ARG1", "`float` :ARG1". The current version of FlowDesigner cannot accept command line arguments with `Object` type such as `Vector<int>` using this method.

See also

None.

7.2 Copying nodes from other network files

Problem

I cannot copy and paste a node from other files.

Solution

This problem often occurs when trying to copy and paste a node from one to another network file. Copy and paste cannot be performed because the two network files were started in different processes.

Copy and paste can be enabled by the following procedure.

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

2) Select the block to be copied, and click `"Edit" -> "Copy "`

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

Also, you can use the following shortcuts:

- `Ctrl-c` : Copy
- `Ctrl-x` : Cut
- `Ctrl-v` : Paste

Windows OS does not support these shortcuts.

7.3 Making an iteration using FlowDesigner

Problem

I wish to construct a network that performs iterations like "for loop."

Solution

Use the Iterator subnetwork and Iterator node. For example, if you want a localization result generated by ConstantLocalization to DisplayLocalization 500 times, build a network similar to that in Fig. 7.1. Note that the Iterator node is at New Node → Flow → Iterator .

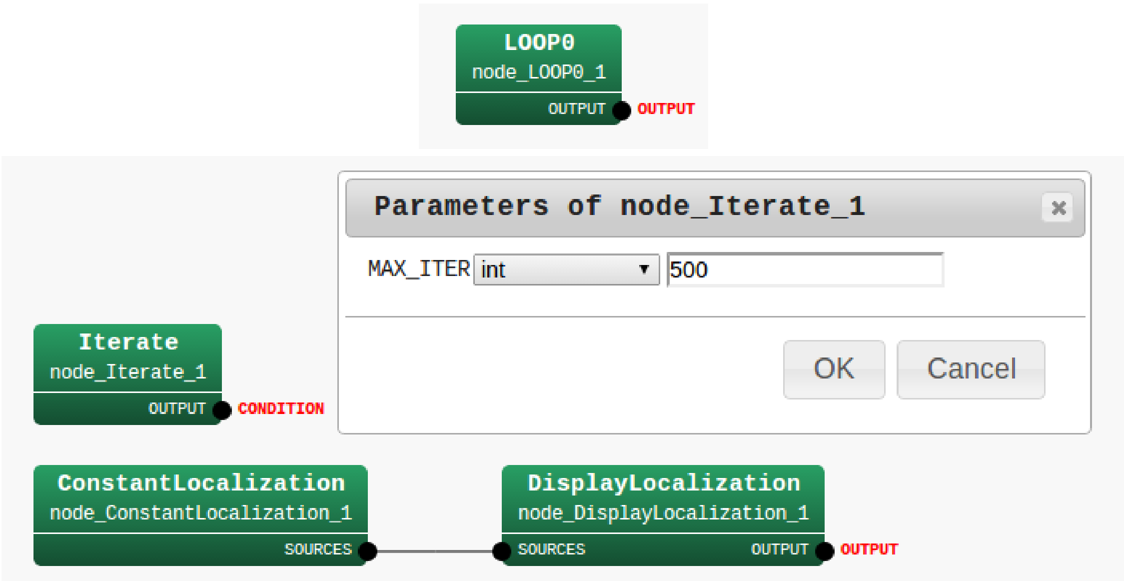


Figure 7.1: Sample network: The left panels is the MAIN subnetwork, and the right panel is the Iterator subnetwork

If you see the localization result for 500 frames, you have successfully built this sample network.

Discussion

Iterate node decrements the counter, whose initial value is given by the parameter MAX_ITER, for each iteration. If the counter is ≥ 0 , it outputs `true`; otherwise, it outputs `false`. By setting this output as the CONDITION terminal, you can realize the "for loop."

See Also

FlowDesigner: Run FlowDesigner , click "Help", and then "User Guide."

Chapter 8

Sound source localization

8.1 Introduction

Problem

I wish to localize sound using a microphone array.

Solution

HARK provides not only sound source localization, but also tracking, visualization, saving, and loading localized results. This recipe introduces other recipes in this chapter. If this is the first time you are performing localization, read Learning sound localization and build a localization system. Then, debug the system using the recipe: Checking if the sound source localization works successfully. For any problems, see recipes such as Too many localization results / no localization results or The localization result is fragmented / Isolated sounds are connected. For problems connecting localization with separation, see Ignoring the beginning of separated sound. To analyze localization results, save them to files using the recipe Saving the localization results to a file

To improve the localization performance, tune the parameters using the recipe Tuning the parameters of sound source localization If your situation includes multiple sound sources, tune the parameters using the recipe Localizing multiple sounds. If you want to localize not only the azimuth but the elevation, see the recipe Localizing the height or the distance of the source. If you want to use part of a microphone array, use the recipe Use the part of microphone array.

Discussion

There are seven primary nodes for localization:

Localization LocalizeMUSIC
Main node. This outputs localization results from input signals.

Generate constant localization ConstantLocalization
Debugging node. This generates constant localization results.

Tracking the localization SourceTracker
This tracks the localization results and gives the same ID to the same source.

Visualize the result DisplayLocalization
Visualization node.

Save and load the result SaveSourceLocation and LoadSourceLocation
This node saves and loads the localization results to a file.

Extend the result SourceIntervalExtender
This node extends the localization results for sound separation.

See also

See the HARK document for the usage of each node. See LocalizeMUSIC in the HARK document for a theoretical description of localization.

8.2 Tuning parameters of sound source localization

Problem

How should I adjust the parameters when sound source localization is suboptimal?

Solution

The solution is given for each sound localization problem.

Q.1) Localization directions are indicated poorly messily or are not indicated at all.

When displaying a localization result in DisplayLocalization , the localization directions may not be are indicated precisely messily in some cases. This is due to the use of a low power source of sound has been localized as a sound source. If no directions are shown When results are not indicated at all, it is because of the opposite reason.

A.1-1) **Change `THRESH` of SourceTracker**
This is the parameter that directly changes the expected threshold value of the direction of a sound source. It should be adjusted so that only the peak of the sound source is captured well.

A.1-2) **Make `NUM_SOURCE` of LocalizeMUSIC equal to the number of target sounds**
This enhances the peak of the target sound direction with NULL space, so that the number of peaks to be enhanced changes according to the setting of `NUM_SOURCE` (number of sound sources). When this setting is wrong, the performance deteriorates, including localizing a peak in the direction of noise or no peak in the direction of the target (In actual localization, only the sharpness of the peaks is degraded, so they still can be used for localization.) If there is only one speaker, the performance will be improved by setting `NUM_SOURCE`1.

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

A.2-1) **Change `MIN_SRC_INTERVAL`in SourceTracker** when there are sound sources nearby (e.g. two sound sources only 10 degrees away from each other). It may be necessary to set the value of `MIN_SRC_INTERVAL` sufficiently small (less than 10 degrees in this example). When the set value is greater than the angle difference, the two sound sources are localized as one sound source.

A.2-2) **Make `NUM_SOURCE`of LocalizeMUSIC equal to the number of the target sounds** Same as A.2-1). Note that if the volume is loud enough, and the sounds are far enough apart (more than 40 deg), localization is usually sufficient well even if the parameter is ill configured.

Q.3) Non-vocal sound is used
`LOWER_BOUND_FREQUENCY`and `UPPER_BOUND_FREQUENCY`of LocalizeMUSIC
Sound source localization is processed for each frequency bin designated for these two frequencies. Therefore, setting a frequency totally different from that of the target sound will result in a wider peak. Use frequencies that correspond to those of target sound sources.

Q.4) I can assume that the sound does NOT come from a certain range.
`MIN_DEG`and `MAX_DEG`of LocalizeMUSIC
The 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.

Discussion

The solutions are parameter tuning of sound source localization. However, if the reverberation of your room is significantly different from the one of which you record the transfer function, you need to re-measure the transfer function. See HARK web page for the transfer function measurement instruction video.

For tuning the sound source localization parameters, especially the ones of SourceTracker node, visualization of MUSIC spectrum is very helpful. Here we describe an example to visualize the MUSIC spectrum using matplotlib, which is a python module.

Step 1: output MUSIC spectrum If you run your network file that includes LocalizeMUSIC node with DEBUG property is `true`, you will see that the output to the console (stdout) includes the lines starts with `MUSIC spectrum`. These values called MUSIC spectrum contain information used by SourceTracker .

Since these values get higher if the sound comes from the corresponding direction at the corresponding time frame, you can see when and from which direction the sound is detected by checking the MUSIC spectrum.

An example of console output is following:

```
reading A matrix
done
0:   17.68 -130.00    0.95   27.71
MUSIC spectrum: 26.409233 26.342979 26.311218 26.389189 26.684574 26.641804 26.473591 26.429607 26
0:   17.68 -130.00    0.95   28.03
MUSIC spectrum: 26.710100 26.621223 26.543722 26.562099 26.749601 26.577915 26.392643 26.393244 26
```

Step2: visualizing MUSIC spectrum If you use `imshow` method in `matplotlib` module, you can easily show the music spectrum.

Assume that you saved the log file above as `log.txt` and the script below as `showMusic.py`.

```
#!/usr/bin/env python
import pylab
import sys

musicspec = [map(float, line.split()[2:]) for line in open(sys.argv[1])
             if "MUSIC spectrum" in line]
musicspec = pylab.array(musicspec).transpose()

pylab.imshow(musicspec, interpolation="nearest", aspect="auto")
pylab.colorbar()
pylab.ylabel("Direction of Arrival")
pylab.xlabel("Time [frame]")
pylab.show()
```

Then, you can see the visualized image of MUSIC spectrum with the following command:

```
$ python showMusic.py log.txt
```

See Also

See LocalizeMUSIC in the HARK document for a detailed description of the MUSIC algorithm and parameters.

8.3 Using part of a microphone array

Problem

I wish to localize a sound source using only part of my microphone array.

Solution

You may want to evaluate your microphone array by using only part of it, e.g., use only 4 channels of an 8 microphone array.

Use the SELECTOR parameter of ChannelSelector for this purpose. Set the type of SELECTOR as `Object` , and the indices of the microphone using `Vector<int>` .

Adjust the SELECTOR parameter of the ChannelSelector module. 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>` . For example, if you want to use only channels 1 and 3 of a 4 channel microphone array, the parameter should be `<Vector<int> 0 2>` .

Discussion

None.

See Also

See ChannelSelector of the HARK document for a more detailed explanation.

8.4 Localizing multiple sounds

Problem

Read this section when

- Localization performance is degraded by the simultaneous presence of multiple sounds, or
- The user wishes to know the values of the properties of the number of sound sources in the LocalizeMUSIC module.

Solution

As a simplification, the number of sound sources that can be localized simultaneously is equal to or less than the number of microphones to be used. If the number of target sound sources to be localized is N , this value should be set at N . If M noises are heard continuously from a specific direction, then this value should be set at $M + N$.

Discussion

Theoretically, the MUSIC method can localize sound sources whose number is equal to or less than the number of microphones. That is, when using eight microphones, the maximum number of sound source is eight. Experimentally, however, stable localization can be performed only for up to $3 \sim 4$ sound sources.

8.5 Checking if sound source localization is successful

Problem

Read this section if

- You wish to confirm that the LocalizeMUSIC module was successful in performing sound source localization, or
- When processing using localization results, such as sound source separation with the GHDSS module, does not perform well.

Solution

Localize a voice and a sound from a speaker in a specific direction and confirm their localization by collating with localization results. To confirm localization results, use the DisplayLocalization module and SaveSourceLocation . To confirm the accuracy of localization, set MIN_DEG of LocalizeMUSIC to - 180 and MAX_DEG to 180 so that sounds from all directions can be localized. If a sound does not come from a specific direction, the localization results may become stable by restricting the direction of the sound source; i.e., by setting MIN_DEG and MAX_DEG appropriately.

Discussion

To improve localization accuracy

1. Measure the transfer functions of the microphone array
2. Appropriately tune the SourceTracker
3. Set so that localization is not performed from an angle at which 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: DisplayLocalization module
- HARK document: SaveSourceLocation module

8.6 Too many localization results / no localization results

Problem

Read this section if

- Localization with the LocalizeMUSIC node is not performed well.
- Despite the absence of sounds, sound sources are continuously localized from a specific direction.
- You wish to set an appropriate value for the THRESH property of the SourceTracker node.

Solution

1. Execute the network file for which the DEBUG property of the LocalizeMUSIC node is set to `true`.
2. Watch the power values of a MUSIC spectrum when there are no sounds and when there are sounds such as clapping.
3. Set the power value to an intermediate between these two.

The power should be set slightly higher than the steady power in the presence of silence. For example, if the steady power is around $25.5 - -25.8$, set THRESH at 26 . In step 1, visualizing the time-direction MUSIC spectrum like a spectrogram facilitates the choice of threshold.

Discussion

Since the values output by the LocalizeMUSIC node are dependent on the gains of the microphones and the surrounding environments, appropriate values should be set by trial and error, as above. The following trade-off relationship arises in the setting of THRESH: When THRESH is set at a small value, localization can be performed for small power sources, allowing localization of unexpected noises (e.g. footsteps). When THRESH is set at a high value, loud sounds are not localized, whereas greater power is needed to localize uttered sounds.

See Also

- Check if the sound source has been localized successfully
- How should I determine PAUSE_LENGTH for SourceTracker ?
- HARK document: LocalizeMUSIC node
- HARK document: SourceTracker node

8.7 Localization results are fragmented / Isolated sounds are connected

Problem

Read this section if

- Despite sounds being continuous, localization results are discontinuous.
- All acoustic localization results are output continuously.
- The user wishes to set an appropriate value for the PAUSE_LENGTH property of the SourceTracker module.

Solution

1. Connect the SourceTracker module to the DisplayLocalization module and display the localization results.
2. Read an appropriate sentence aloud, and see the localization results.
3. **Localization results break off:** Increase the value of PAUSE_LENGTH.
4. **Localization results are too close:**

Discussion

The purpose of the PAUSE_LENGTH property is to recognize speech appropriately, even if the power of a MUSIC spectrum in the LocalizeMUSIC module localizes it as continuous speech. Since this is applicable only to human speech, such a sound can be used. If your purpose is to localize human speech, use the default value.

PAUSE_LENGTH units PAUSE_LENGTH is measured in milliseconds. Therefore, the maximum PAUSE_LENGTH depends on the sampling frequencies of the AudioStreamFromMic and AudioStreamFromWave modules (SAMPLING_RATE) and the step size (ADVANCE) of FFT. If all parameters are set at their default settings (sampling frequency, 16000Hz; step size, 160 pt), changing PAUSE_LENGTH by 1 corresponds to changing it 1 msec.

See Also

- How should I use SourceIntervalExtender ?
- HARK document: SourceTracker module

8.8 The beginning of the separated sound is ignored

Problem

Read this section if

- The beginning part of the separated sound breaks off.
- The beginning silent section of the separated sound is too long.
- The user does not know how to use the SourceIntervalExtender module.

Solution

Adjust as follows.

1. Create a network file that can save a separation result (see How should I save separated sounds in files?). In this case, sandwich the SourceTracker and SourceIntervalExtender modules between a localization module such as the LocalizeMUSIC module and a separation module such as GHDSS module.
2. Separate a sound and display or listen to the result.
3. **If the beginning part of the separated sound breaks off**, increase PREROLL_LENGTH
4. **If the beginning silent section of the separated sound is too long**, reduce PREROLL_LENGTH

Discussion

At the time point sound source localization is first reported, 500 msec has already elapsed from the start of the utterance and the beginning part of the separated sound is lost, leading to a failure of speech recognition. The SourceIntervalExtender module is designed to solve this problem. Measure PREROLL_LENGTH to determine how far to trace back from the start of sound source localization and separation. If PREROLL_LENGTH is too low, the beginning part of a separated sound will be lost, affecting on speech recognition. If, however, PREROLL_LENGTH is too high, an utterance may be connected to the one before or after it, leading to recognition errors in some language models used for speech recognition.

Unit of PREROLL_LENGTH The unit of PREROLL_LENGTH corresponds to 1 time frame when performing a Fourier transform for a short time. Therefore, its correspondence to actual time depends on the sampling frequency (SAMPLING RATE) designated for the AudioStreamFromMic and AudioStreamFromWave modules and the step size (ADVANCE) of FFT. If all are set at their default settings (sampling frequency, 16000Hz; step size, 160 pt), a change of PREROLL_LENGTH of 1 corresponds to a change of 10 msec.

See Also

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

8.9 Localizing the height or distance of a source

Problem

Read this section if you wish to determine

- The height of a sound source, as well as its direction on a horizontal surface
- The distance to a sound source as well as the direction it is coming from.

Solution

The current LocalizeMUSIC module estimates only the direction of sound on a horizontal surface. To also estimate the height of a sound source and its distance from a microphone array, the program of the module must be remodeled. Since the MUSIC algorithm itself does not assume angles on a horizontal surface, it may be expanded. However, transfer functions for each datum to be localized will be required. For example, to estimate the height of a sound source, a transfer function from the sound source is required when changing the horizontal direction and height. Moreover, the microphone array must be positioned to capture the required information. For example, for direction localization on a horizontal surface, the microphones should be positioned on a horizontal surface; for also estimating height, the microphones should be positioned on the surface of a sphere.

Discussion

Since the MUSIC algorithm estimates source locations based on prior transfer functions, this algorithm can be used to estimate height by measuring a transfer function according to the information required. However, since the implementation in HARK is limited to determining the direction of a sound source on horizontal surfaces, appropriate modifications are required.

See Also

- HARK document: LocalizeMUSIC

8.10 Saving the localization results to a file

Problem

Read this section if

- You do not know how to save localization results in files.

Solution

Connect the SaveSourceLocation module to a module that outputs localization results such as the Localize-MUSIC , ConstantLocalization and LoadSourceLocation modules. Designate a name of a file where localization results are saved in the FILENAME parameter.

Discussion

None.

See Also

- HARK document: SaveSourceLocation

Chapter 9

Sound Source Separation

9.1 Introduction

Problem

I want to separate a mixture of sounds.

Solution

HARK provides a sound separation node GHDSS to separate a mixture of sounds recorded by a microphone array. Sound must be localized first, because GHDSS inputs the direction of the sound. See Learning sound separation to build a sound source separation system.

GHDSS requires a transfer function from the position of the sound to each microphone. Two types of transfer function can be used: (1) actual measurements using TSP (Time Stretched Pulse) and (2) calculations from the microphone configuration by simulation.

1. Transfer function from recording
   Put the microphone in a room and record the TSP signal. This improves separation performance by providing an actual transfer function. See Recording impulse response.

2. Transfer function from simulation
   Instead of recording impulse responses, you can calculate the transfer function from the locations of each microphone (the file is called MICARY-LocationFile). See harktool in the HARK document, and Sound source separation using only the microphone array layout to determine how to make the MICARY-LocationFile

After determining the transfer function the parameters must be configured (see Learning sound separation).

To save the separated sound itself, see the recipe Saving separated sounds to files. To improve the separation performance, see the recipe: Tuning the parameters of sound source separation.

To use the system in a noisy environment, see the recipe Reducing the leak noise by post processing.

If your target sound or microphone array moves, see the recipe Separating the moving sound.

Discussion

GHDSS separates sound using a higher-order decorrelation and geometric constraints based on the sound. See GHDSS module in the HARK document for details.

See Also

The node descriptions in the HARK document and the materials in the HARK tutorial may be helpful.

9.2 Saving separated sounds to files

Problem

Use this section when listening to separated sounds for checking or when saving a separated sound to a file for experiments.

Solution

Use the SaveRawPCM or SaveWavePCM module. The output format of SaveRawPCM module is the Raw format of Integer with no headers. The output format of SaveWavePCM module is the wave format of Integer with a header. Saving methods differ according to the values to be saved.

1. Saving real number signals (temporal waveforms)

When saving temporal waveforms, connect SaveRawPCM or SaveWavePCM , as shown in Fig. 9.1. If not connecting a module that performs spectral transforms, such as MultiFFT , set the ADVANCE parameter identical to the number of dimensions of the input object (INPUT).

2. Saving complex signals (spectrum)

When saving spectra, connect SaveRawPCM or SaveWavePCM as shown in Fig. 9.2. Both must first be resynthesized to form temporal waveforms. Connect the Synthesize module and convert the spectra into temporal waveforms. Subsequently connect SaveRawPCM or SaveWavePCM , as shown above for real number signals. Here, the ADVANCE parameter must be same as that of the Synthesize module. Determine if the signals have been saved successfully by seeing if a separated sound file was generated at the time of execution.

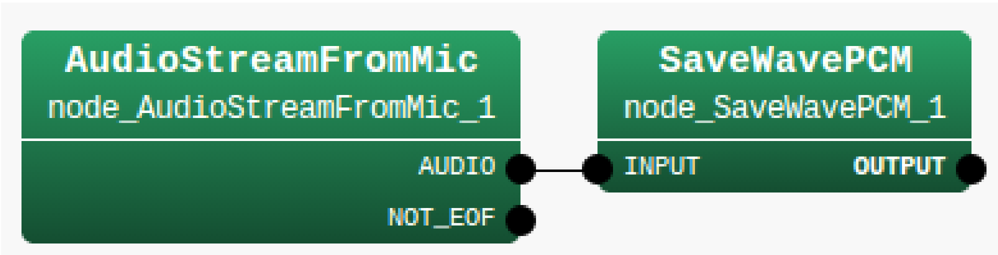


Figure 9.1: Connection example1

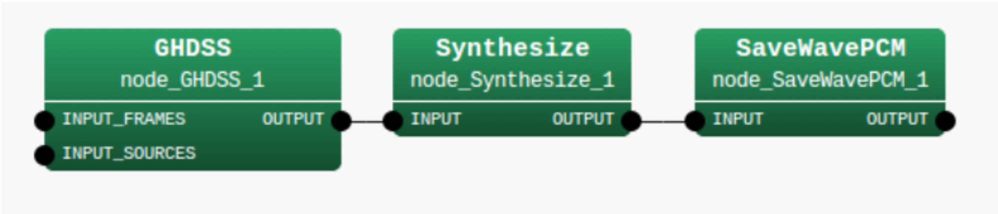


Figure 9.2: Connection example2

Discussion

The SaveRawPCM and SaveWavePCM modules differ only in their headers. In general, the wave format has a header making is easier to deal with conventional software. Therefore, users should use the SaveWavePCM module.

See Also

Module references for Synthesize , SaveRawPCM and SaveWavePCM .

9.3 Parameter tuning of sound source separation

Problem

How should I adjust its parameters when sound source separation is suboptimal?

Solution

This section describes the settings for GHDSS , the primary module for sound source separation.

1) **Settings dependent on pre-measured spatial transfer function**

Since GHDSS separates using information on spatial transfer functions measured beforehand or calculated from microphone positions, its settings must be in accordance with those of the transfer functions concerned. Concretely, they correspond to the following set values.

- `LC_CONST = FULL`
  When transfer functions are measured properly, set this to `FULL`. Otherwise, `DIAG`.

2) **Determination of the curvature of non-linear constraints**

In GHDSS , the coefficient `SS_SCAL`, corresponding to the curvature (gradient at the origin) of a sigmoidal function, determines the performance. Increasing this curvature brings it closer to the linear constraint, whereas decreasing this curvature increases its non-linearity. Since too low a setting would result in a dull adaptation, its value should depend on the target environment.

3) **Initial separation matrix**

You can specify the initial separation matrix by setting `INITW_FILENAME`. An appropriate initial separation matrix reduces the convergence time for the separation. If you do not specify a file name, the initial matrix is constructed by the transfer function file and the localization result from LocalizeMUSIC .

4) **Step size calculation method**

There are two types of step size calculation methods, `SS_METHOD` for higher-order decorrelation, and `LC_METHOD` for geometric constraints. The performance of both is improved by setting them to `ADAPTIVE` , unless the environment has been highly optimized. If you set `SS_METHOD = LC_METHOD = FIX`, `SS_MYU = LC_MYU = 0` and input an initial separation matrix by `INITW_FILENAME`, separation may be realized by fixed beamformer. The details of each parameter are shown below:

4-1) **Step size calculation method for higher-order decorrelation :** `SS_METHOD`

1. FIX
If `SS_METHOD` is set at `FIX`, `SS_MYU` will appear in the property window. You can input a value, e.g., 0.001, in `SS_MYU`. A larger value reduces convergence time, while decreasing stability and accuracy. In contrast, a smaller value improves convergence stability and accuracy, while decreasing convergence time.

2. ADAPTIVE
If `SS_METHOD` is set as `ADAPTIVE`, the step size will be automatically optimized, improving the stability and accuracy of convergence.

3. LC_MYU
In this case, step size is defined by the `LC_METHOD`.

4-2) **Step size calculation method for geometric constraints :** `LC_METHOD`

1. FIX
If you set LC_METHOD as FIX, you can see LC_MYU. The description is the same as that for SS_METHOD.
2. ADAPTIVE
If LC_METHOD is set to ADAPTIVE, the step size will be automatically optimized.

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)” in the HARK training session material.

9.4 Sound source separation using only the microphone array layout

Problem

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

Solution

You obtain the transfer function in simulations with harktool . For details, see the description of the harktool .

Discussion

None.

See Also

harktool

9.5 Reducing noise leakage by post processing

Problem

Use this section when distortion is included in the separated sound and when wishing to improve automatic speech recognition by speech enhancement.

Solution

This section describes the settings of nodes related to speech enhancement: PostFilter , HRLE , WhiteNoiseAdder and MFMGeneration .

1) **PostFilter**

Depending on the situation, better recognition performance is obtained without PostFilter . It is necessary to set adequately the parameters of PostFilter for the given environment. Since the default parameters are determined based on the environment used by the HARK development team, there is no guarantee that they will be suited to the user's environment.

PostFilter contains many parameters, with many being interdependent. Therefore, it is extremely difficult to tune by hand operations. One solution is to use a 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 for evaluations. Note that the system may learn parameters too specialized for the given environment.

In PostFilter , stationary noise, reverberation and noise leakage are dynamically estimated by the magnitude relationships of input signal power, with more precisely separated sounds obtained by subtraction. Under some conditions, performance may be degraded because the speech is distorted by such subtraction. Therefore, PostFilter is affected by estimations of stationary noise, reverberations and noise leakage. The influence of PostFilter can be minimized by setting the following parameters to 0.

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

To increase the influence of PostFilter , bring these values closer to 1.

2) **HRLE**

The number of parameters is much smaller in HRLE than in PostFilter . HRLE can enhance speech by calculating the spectral histograms of separated speech signals and detecting differences between noise and speech. Therefore, the design of the histogram has marked effects on speech enhancement performance. HRLE includes 5 parameters: `LX`, `TIME_CONSTANT`, `NUM_BIN`, `MIN_LEVEL`, and `STEP_LEVEL`. All these parameters, except for `LX`, are appropriate in the default setting. However, since `LX` defines the level of the surface between noise and speech, the best value depends on each acoustic environment. A higher `LX` can suppress high power noise but increase acoustic distortion. In contrast, a lower `LX` will reduce the distortion, but not suppress high power noise. Thus, set an appropriate `LX` depending on your environment.

3) **WhiteNoiseAdder**

Adjust the value of `WN_LEVEL`. If it is too small, the distortion generated in the separated sound cannot be subtracted sufficiently. If it is too large, not only the distorted part but also the separated sound itself will be affected due to too much subtraction.

4) **MFMGeneration**

Threshold values to mask features can be changed by changing the `THRESHOLD` value in the range from 0 to 1. All features are not masked when `THRESHOLD` is 0, indicating that some unreliable features are used for speech recognition. As `THRESHOLD` gets closer to 1, all features are masked, indicating that all features are not used. Both too high and too low values of `THRESHOLD` would degrade speech recognition.

Discussion

None.

See Also

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

9.6 Separating a moving sound.

Problem

Use this section if you do not know how to set UPDATE_METHOD_TF_CONJ and UPDATE_METHOD_W as properties of GHDSS .

Solution

If you do not understand well, use the default value. When changing it, you must determine if you are focusing on the sound source itself (ID) or the direction (POS). In particular, when a sound source moves or a robot's body moves, separation results change under conditions in which good separation traceability is needed. When focusing on ID, reuse the ID from an earlier step as a geometric constraint and separation matrix value. Therefore, when a sound source moves fast, those values will become inappropriate and separation will be unsuccessful. In contrast, when a sound source does not move much, the separation accuracy will continue to improve during reuse. When focusing on POS, the characteristics are reversed compared with ID. A focus on POS is suited for when a sound source moves at high speed. When the interval between movements is large, a database is used for values of a constraint condition, and the separation matrix or system is initialized. An ideal way is to apply these constraints to each sound source dynamically. For details, see the description of the GHDSS module.

1. UPDATE_METHOD_TF_CONJ

2. UPDATE_METHOD_W

Discussion

None.

See Also

GHDSS

Chapter 10

Feature Extraction

10.1 Introduction

Problem

Read this section to learn about the features available for speech recognition.

Solution

The features used for common speech recognition include:

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)

HARK supports only MFCC and MSLS. For speech recognition using an acoustic model distributed on the web, use MFCC. For speech recognition based on missing feature theory, MSLS is better than MFCC.

Discussion

The LPC coefficient is a parameter of a model of a spectrum envelope. It is based on the value at time of t in the stationary process x_t being correlated with that of a recent sample. Figure 10.1 shows how to obtain the LPC coefficient. The LPC coefficient is a prediction coefficient (a_m), in which 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 are minimal. Since this LPC yields a comparatively precise speech model, it has been used widely for speech analysis-synthesis. However, a model based on LPC has a high coefficient of sensitivity and may become unstable due to a slight error in this coefficient. Therefore, speech analysis-synthesis is performed in the form of PARCOR.

PARCOR is a correlation coefficient of prediction errors of x_t (forward) predicted from $x_{t-(m-1)}, \dots, x_{t-1}$ and x_{t-m} (backward). Figure 10.2 shows how to derive this PARCOR. In principle, a model based on this PARCOR is stable [1] .

MFCC is a cepstrum parameter and is derived with filter banks placed at even intervals on a Mel frequency axis[1] . Figure 10.3 shows its process of derivation.

MSLS is derived by a filter bank analysis similar to that of MFCC, without performing the reverse discrete cosine transformation, the final step of MFCC extraction processing, and is a feature remaining in the frequency domain. When the noise at a specific frequency is mixed with acoustic signals, specific features including the frequency are affected in MSLS. For MFCC, however, the influence of noise spreads and many features are affected. Therefore, in general, MSLS performs well when combining the missing feature theory for speech recognition.

[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.

See Also

MFCCExtraction and MSLSExtraction in HARK Document.

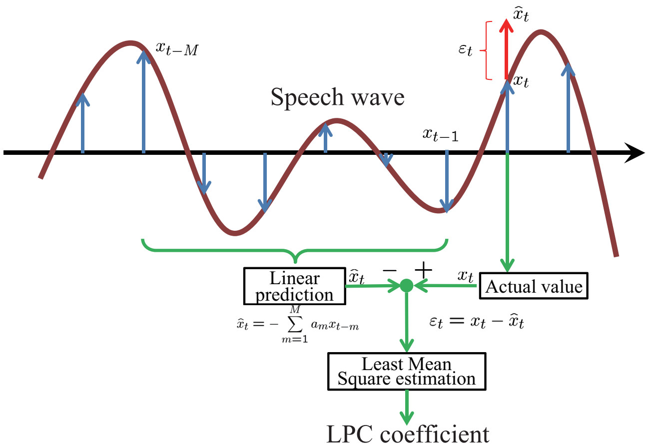


Figure 10.1: LPC coefficients

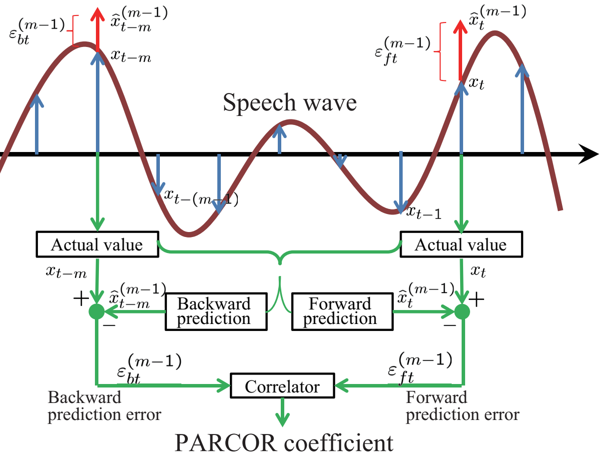


Figure 10.2: PARCOR coefficients

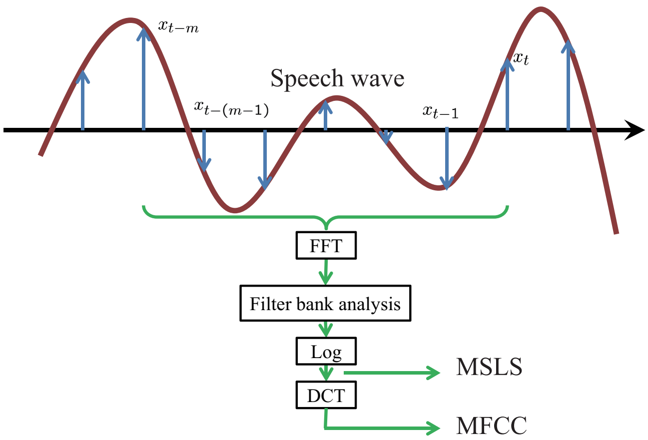


Figure 10.3: MFCC and MSLS

10.2 Selecting the threshold for Missing Feature Mask

Problem

Read this section if you do not know how to set parameters of the MFMGeneration module.

Solution

MFMGeneration includes the parameter THRESHOLD, which affects the performance of speech recognition. If the threshold value is set to 0.0, speech recognition will not be based on the missing feature theory. If it is set to 1.0, all features are covered with masks and therefore recognition is performed without any features. A suitable value is obtained experimentally through actual recognition, by changing threshold values in increments of 0.1.

Discussion

MFMGeneration is 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) (hard mask).

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

where f ,p, $m(f,p)$, and $r(p)$ represent the frame, dimension, mask, and reliability of a feature, respectively.

See Also

MFMGeneration in HARK Document

10.3 Saving features to files

Problem

Read this section if you wish to save features extracted with HARK.

Solution

Use SaveFeatures or SaveHTKFeatures module to save features. The SaveFeatures module saves features in float binary format, whereas the SaveHTKFeatures module saves features in HTK format. Figure 10.4 shows a network example, in which features are saved. Here, features are extracted and saved from a 1 channel audio signal read from the AudioStreamFromMic module. Features are saved by assuming the extracted features as inputs of the SaveFeatures module.

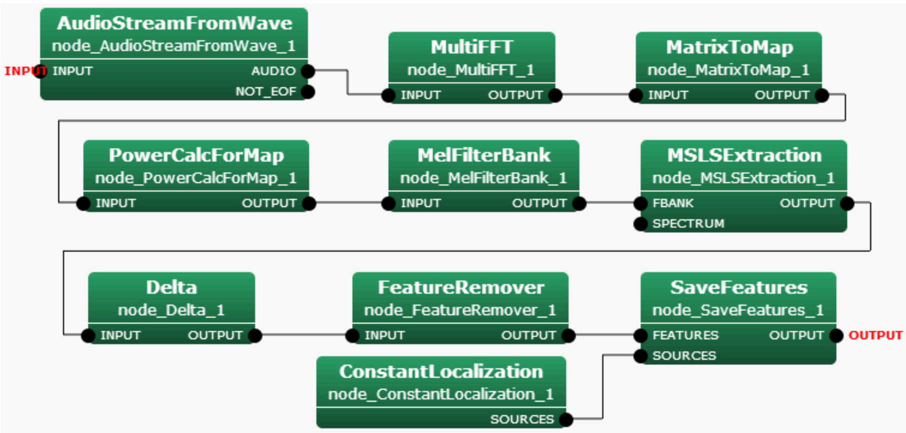


Figure 10.4: Sample network for saving features

Discussion

The format used in the SaveFeatures module is 32 bit float and little endian, and the format used in the SaveHTKFeatures modules is HTK. The HTK format is better for training an acoustic model using HTK.

See Also

SaveFeatures and SaveHTKFeatures modules and file format in HARK Document

Chapter 11

Speech Recognition

11.1 Making a Julius configuration file (.jconf)

Problem

Read this section if you do not know how to make a Julius configuration file (.jconf file) and what options should be set.

Solution

Julius includes many optional parameters that can be set by the user. It is complicated to designate options in Julius every time using command lines. A series of optional parameters can be set in a text file, thus simplifying option inputs in Julius. This text file is called a .jconf file. The options used in Julius are summarized in http://julius.sourceforge.jp/juliusbook/ja/desc option.html. All options can be described in text in a .jconf file. Important reminders when connecting Julius to SpeechRecognitionClient (or SpeechRecognitionSMNClient) are summarized. The minimum setting items are

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

-notypecheck is an essential option. HARK uses an expanded acoustic parameter structure and 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 the type check of features and does not recognize sounds.

For -plugindir, designate the path in which function enhancement plug-in files such as mfcnet are saved. This path must be designated before -input mfcnet and -gprune add masks. All enhancement plugs in files present in the plug-in path are read. For Windows OS, this option must be set even when the input option is not mfcnet. If mfcnet is disabled, the directory name can be arbitrary.

-input mfcnet is an option to recognize features received from SpeechRecognitionClient (or SpeechRecognitionSMNClient). Designate this option to support missing feature masks. Select the Gaussianpruning algorithm to support -gprune. Since calculations are performed while supporting the missing feature masks, the option name is different from that for normal Julius. Select from $\{add\_mask\_to\_safe\ ||add\_mask\_to\_heu$

$||add\_mask\_to\_beam\ ||add\_mask\_to\_none\}$, which correspond to $\{safe\ ||heuristic\ ||beam||none\}$ of normal Julius, respectively.

In addition, it is necessary to designate a language model and an acoustic model, the same as for normal speech recognition. Designate a grammar file in -gram, a definition file in -h HMM and an HMMList file in -hlist.

Chapter 12

Others

12.1 Selecting window length and shift length

Problem

Read this section to determine optimal window and shift lengths for analyses.

Solution

Length is the window length of speech for analysis, generally 20-40 ms. If the sampling frequency is fs Hz, length = fs/1000 * x, with x being 20-40 ms. Advance is an analysis frame shift length, which generally overlaps 1/2-1/3 of the preceding and following frames. When performing speech recognition, it is necessary to use the same Length and Advance for acoustic model creation.

Discussion

When analyzing speech, the range in which signals can be assumed to be weakly stationary is 20-40 ms; therefore, this section describes settings yielding these lengths. Shift length is determined as the execution width of a window. Concretely, determine the head of a rectangular window with energy equivalent to that of a window function. This window length is not utilized for frame processing of the same sample redundantly when analyzing continuous frames, making frame processing possible without discarding samples. Since the energy of window functions for speech analyses is about 1/3-1/2 of the rectangular window length, the amount of frame shift should be within this range. Although 1/3 is a conservative setting and may cause redundant frame processing of the same sample, few samples are discarded. Although samples may be discarded at settings of 1/2, depending on window functions, redundant frame processing does not occur. However, when using a rectangular window for analysis, the shift length must be equal to the analysis frame length. For triangular windows, the frame shift amount is 1/2.

12.2 Selecting the window function for MultiFFT

Problem

Read this section to determine how to choose a window for MultiFFT .

Solution

(three kinds: HUMMING, CONJ and RECTANGLE) For speech analysis, choose HUMMING. For other signals, choose an appropriate window for spectral analysis of signals.

12.3 Using PreEmphasis

Problem

Read this section if you do not know whether to use the time domain or frequency domain for PreEmphasis .

Solution

The necessity and effects of PreEmphasis for general speech recognition have been described in various books and papers. PreEmphasis can be used in both a time and a frequency domain. However, it is better to choose a domain while considering the data used for acoustic model training.

Chapter 13

Advanced recipes

13.1 Creating a node

Problem

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

Solution

To create a new node, it is necessary to install HARK by compiling a source, not by a debian package. To install from a source compilation, see "Installation of HARK" in the HARK training session material. When it is ready, describe the source of the node to be created. To learn how to make a basic node, consult the following items included in "Creation of a 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 the actual creation of nodes such as `PublisherInt.cc` and `SubscriberInt.cc`

- Addition of input
- Addition of output
- Buffer (`Lookback Lookforward`)
- Input-output of each type
- Switching the number of inputs from static to configurable

Creation of PublisherInt.cc

First, create `PublisherInt.cc`, which reads integers as a parameter and discharges them without change. (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>
```

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

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

Declaration of a name space. Since all the classes of FlowDesigner , the basis of HARK, are defined in the name spaces of FD, make sure to declare them when abbreviating.

```
class PublisherInt;
```

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

```
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*/
```

In DECLARE_NODE, make sure that the PublisherInt class is defined as one node (an error will occur if it is not the 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. Four values are must be set: 1) the main body of the node, 2) the node inputs, 3) the node outputs, and 4) the node internal parameters. Other than the setting of 1), multiple values can be used (the setting method for multiple values are described later). The following are the concrete set values.

- Setting of the main body of the node
  - @name: Node name indicated on FlowDesigner (should be the same as the class name)
  - @category: Setting of the category to which the node belongs when right-clicking on GUI of FlowDesigner .
  - @description: Description of the node (displayed when placing the mouse over the node in FlowDesigner . Can be omitted.)
- Setting of node inputs
  - @input_name: Name of input indicated in the node
  - @input_type: Type of input variable
  - @input_description: Description of the input variable (can be omitted)
- Setting of node outputs
  - @output_name: Name of output indicated in the node
  - @output_type: Type of output variable
  - @output_description: Description of the output variable (can be omitted)
- Setting of internal parameter of the node
  - @parameter_name: Name of the parameter indicated in the node (indicated in a yellow window when placing the mouse over it)
  - @parameter_type: Type of parameter
  - @parameter_value: Initial value of the parameter (can be changed in the source)
  - @parameter_description: Description of the parameter (can be omitted).

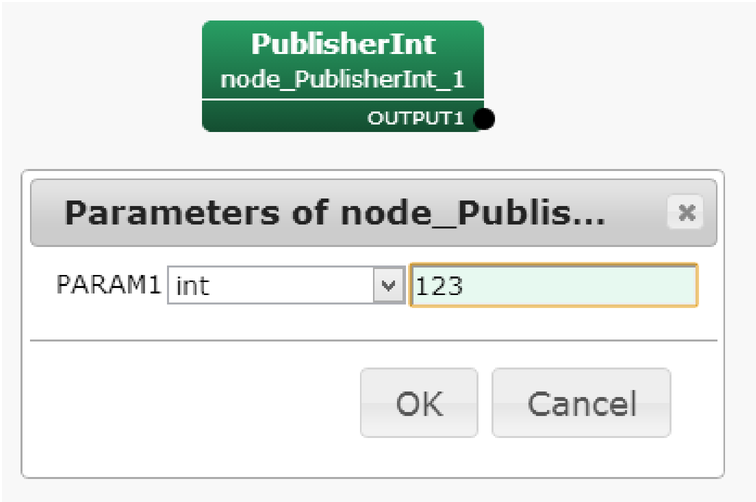


Figure 13.1: PublisherInt node

This source has one output and one internal parameter, and therefore they are displayed as Fig. 13.1 in FlowDesigner .

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

Define the `PublisherInt` class that inherits the `BufferedNode` class, with the latter defined in FlowDesigner .

`outputID` is an integer that stores an ID of an output port. The pointer to be passed to the output port is obtained 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;
  }
```

The constructer that inherits the `BufferedNode` class. `nodeName` (the class object name in the network files of FlowDesigner) and `params` (an initializer of the variable `parameters` contained within the `Node` class and with internal parameters defined for some) are used as arguments.

`output1ID = addOutput("OUTPUT1");` becomes a row that stores the ID of `OUTPUT1` set in the FlowDesigner GUI in `output1ID` defined in the class.

`param1 = dereference_cast<int>(parameters.get("PARAM1"));` is the internal parameter set in the FlowDesigner GUI cast into `int` type. `@parameter_types` include `int` type, `float` type, `bool` type and `string` type, with others called Objects. (`string` type is called an `Object` and is cast into `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"));`

`String` is not `std::string` and `Vector` is not `std::vector` because these types are special types for inputs and outputs of FlowDesigner . Errors occur if information is not transferred in these types. When setting `inOrder = true;`, `count` value increases by one every time `calculate` is performed (details below). In further describing 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.
}
```

This is the main routine of the node. Its content is calculated repeatedly for each `count`, making it an argument of the loop.

In this node, the value of `PARAM1` is only passed to the next and therefore only the information of a current loop is required. To calculate an average value over plural loops, it is necessary to have buffers for several frames. 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 specified by `count`-th "`output1ID`" is regulated.) Since this node is has one output, the output can be expressed as `(out[count] = ObjectRef(Int:: alloc(output1)` although it is expressed as above in general cases. (In the case of one output, `*(outputs[output1ID]. buffer)` is equivalent to `out`.)

`output1`, of `int` type, is cast into `Int` type, making all variable types related to inputs and outputs of `Int` type, `Float` type, `String` type, `Bool` type, `Vector` type, `Matrix` type and `Map` type, the unique types for 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. Inclusion 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. Includsion of `Matrix.h` is required)

Here, RCPtr is an object smart pointer for FlowDesigner . This pointer is passed for inputs and outputs of arrays such as Matrix and Vector .

Install PublisherInt.cc

Compile the source and install it so that PublisherInt.cc can be used in FlowDesigner . First, add

```
PublisherInt.cc \
```

to 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/
```

set

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

and install it. (For {$install dir}, follow your own setting; e.g. /usr).

Start FlowDesigner .

```
> flowdesigner
```

When GUI starts, confirm if there is a node created by

```
Right-click > HARK\_TEST > PublishInt
```

Now the installation is completed. The following shows trouble shooting steps to perform if the above are not displayed.

- Confirm that the directory designated in ./configure --prefix=/*** is the same as that designated in
  flowdesigner-0.9.1-hark.
  Since FlowDesigner reads the def file in its own installed directory, it ignores this file when it is present in other directories.
- Confirm that the script that compiles the node has been created in {$PACKAGE}/hark_test/src/Makefile. Confirm that autoreconf has been performed properly and that Makefile has been rewritten properly.
- Confirm that the node name is same as the class name in the source of the cc file. If they are not the same, they may be compiled but not displayed in GUI.
- Confirm that the path setting is correct ($ which flowdesiner). This problem may arise if a user had previously installed FlowDesigner and HARK in /usr/bin by the root authority and has now installed it locally.

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 into {$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`, we will focus on the differences.

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

Although `PublisherInt.cc` does not have an input port, an input requires that GUI of FlowDesigner be set first. Its format is basically the same as that of `@output`. Since `SubscriberInt.cc` has one input, the following is indicated in FlowDesigner .

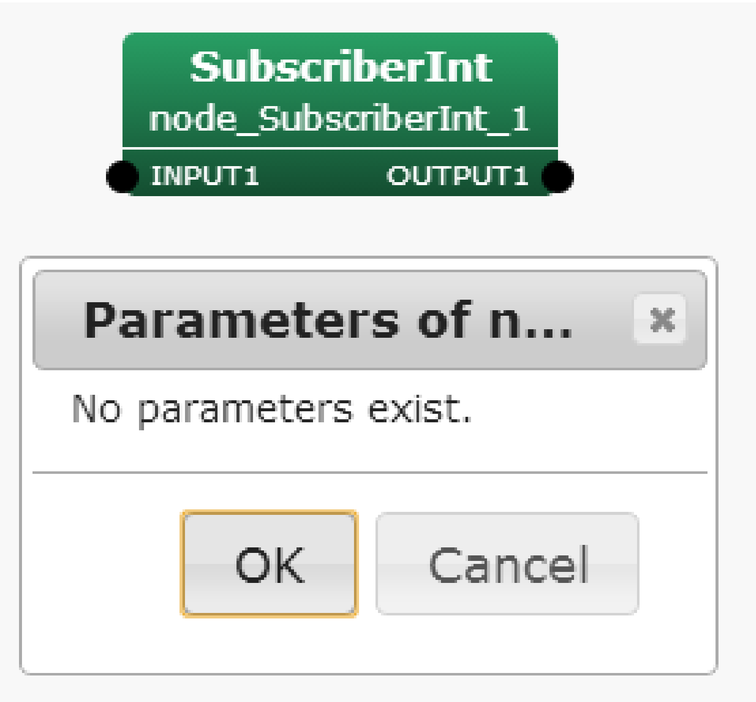


Figure 13.2: SubscriberInt node

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

This code registers the ID `INPUT1` set in the GUI as `input1ID` defined in the cc file and allocates the GUI port to input data.

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

Data are received from the input port corresponding to its ID as an original type of FlowDesigner and are cast into `int` type. Like `ObjectRef` (Int type, Float type, Bool type, String type, etc...), an original FlowDesigner variable type is passed in the output port, with recasting performed in the receiving node in the above process. The following shows examples for other types.

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

The output port setting is same as for `PublisherInt.cc`. When finishing its creation, install it by source compilation using the same procedure as that described in the preceding chapter. Start FlowDesigner and confirm that `SubscriberInt.cc` has been installed properly.

Create network files with PublisherInt.cc and SubscriberInt.cc

Now, we will show how to create a network file (N file) in FlowDesigner using `PublisherInt.cc` and `SubscriberInt.cc`. The network file is show below; if you understand this figure, you do not need to read the rest of this chapter, so proceed to the next chapter.

First, start FlowDesigner. For a new file, only the `MAIN` sheet will appear when starting-up. This is the main process in the program flow.

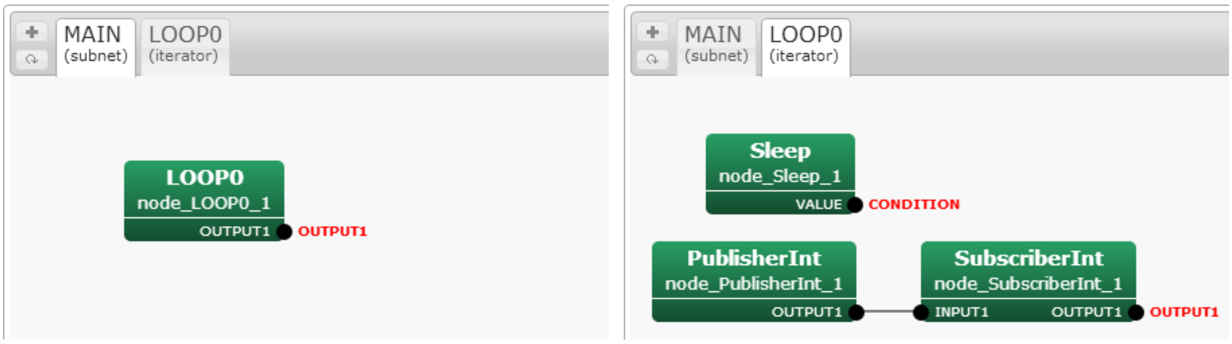
```
Networks > Add Iterator
```

Add a loop processing sheet. Pick a proper sheet name (`LOOP0` here). First, in the `MAIN` sheet side,

```
Right-click > New Node > Subnet > LOOP0
```

so that `LOOP0` can be executed from the `MAIN`. Move to the `LOOP0` sheet and determine the sampling period of repetition processing (both event-based and time-based loops are possible). Here, we perform time-based repeat calculations with `Sleep`.

```
Right-click > New Node > Flow > Sleep (Put the Sleep node in the sheet)
Left-click Sleep > Set parameter \verb|SECONDS| properly (e.g. 10000) > OK
Click the output-terminal of Sleep with pressing "Ctrl"
```



(a) MAIN(subnet) Sheet (b) LOOP0(iterator) Sheet

Figure 13.3: PublisherInt + SubscriberIntnetwork file

Confirm that the output-terminal of `Sleep` has become `CONDITION`. This is a trigger of loop processing, indicating that a new loop begins when the processing of `Sleep` is completed. The main processing is described next.

Put two nodes in the `LOOP0` sheet.

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

```
Left-click PublishInt > Set PARAM1 of PublisherInt (e.g. 123) > OK
Connect the output-terminal of PublisherInt to the input of SubscriberInt
Click the output-terminal of SubscriberInt with pressing "Shift"
```

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

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

Then `OUTPUT1` will become the output from the `MAIN` sheet, enabling `LOOP0` to operate.

```
Save the file using an arbitrary name
Press "Execute"
```

The following output will appear 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 a basic tutorial for creating nodes. Further techniques, such as the addition of inputs and outputs and processing between multiple frames, will be described below.

Adding internal parameters to a node

There is one property parameter called `PARAM1` in `PublisherInt.cc`. This section describes how to change `PublisherInt.cc` to arrange multiple parameters. Since the use of simple `int` type has been described, this section describes how to read `Vector` type, which is more difficult as a parameter (the new parameter is named `PARAM2`). The goal is to read a `Vector` type variable as a parameter, multiply it by `PARAM1` as shown below and modify the node so that the result is output from the output port.

`PARAM1 * PARAM2`

This tutorial involves two elements: 1) addition of an internal parameter in `Vector` type, and 2) addition of an output port in `Vector` type. Therefore, the two are described in two independent chapters. This chapter describes how to add an internal parameter. Open `{$PACKAGE}/hark_test/src/PublisherInt.cc`. Since we are dealing with a `Vector` type variable, include `Vector.h` first.

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

Change the GUI setting 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 * PARAM2
*
```

Here, see `@parameter_value`. The format of `<Vector<int> 0 1 2>` is strict (e.g. spaces). Next, add the following to the member variables of the class.

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

Add the following in the constructor.

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

Then, it can be used as `Vector` , as shown in the following example.

```
for(int i = 0;i < param2.size();i++){cout << param2[i]<< " ";}
```

Although this section described changes only with sentences, the finalized source code is shown in the last part of the next section.

Adding output to a node

The previous section described how to read variables of `Vector` type as internal parameters. In this section, `PARAM1` is multiplied by the `Vector` type variable, and `PublisherInt.cc` is modified so that it can output a Vector type variable. First, add the output for GUI of FlowDesigner:

```
*
* @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;
```

Add the following in the constructor.

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

Set the output in the main routine of `calculate`.

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

Here, since we are calculating `PARAM1 * PARAM2`, the number of components in the output vector is the same as that of `PARAM2`. Therefore, the size of `output2` is the same as that of `param2.size()`. Substitute `PARAM1 * PARAM2` for `output2`.

```
for(int i = 0;i < param2.size();i++){
  (*output2)[i]= param1 * param2[i];
}
```

The results of `PARAM1 * PARAM2` are output from `OUTPUT2`. The finalized edition of `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"));
    param2 = object_cast<Vector<int> >(parameters.get("PARAM2"));
    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));

    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.
  }
};
```

Adding input to a node

We have described a method of outputting a variable of Vector type to the new `PublisherInt.cc`. This chapter describes how to create a new `SubscriberInt.cc` that receives a Vector type variable as an input.

Open {$PACKAGE}/hark_test/src/SubscriberInt.cc using an appropriate editor. Since a Vector type variable is treated, include Vector.h first.

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

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

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

Add a member variable of the class.

```
int input2ID;
```

Add the following to the constructor.

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

Read input data in calculate.

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

Then, input2 can be used in the main routine:

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

The finalized 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 an N file with new PublisherInt.cc and SubscriberInt.cc

The fundamental procedure is same as that for N file creation described in the previous section. First, confirm if compilation and installation were performed properly. If successful, start FlowDesigner . Open the N file created in the previous section. Confirm that: 1) `OUTPUT2` has been added to the output ports of the `PublisherInt` node, and 2) `INPUT2` has been added to the input ports of the `SubsctiberInt` node. (If these cannot be confirmed, installation was not successful, so trouble-shoot the problem as described in the previous section.) After connecting the two, utilize the following setting:

```
Left-click PublishInt > Set type of PARAM2 to "object" > OK
```

Save it 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 ]
...
```

The above is displayed in the console. Confirm that `Vector` is exchanged correctly. The following is the capture of the node created.

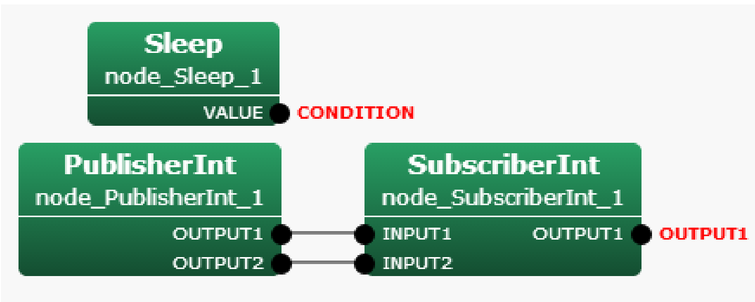


Figure 13.4: SubscriberInt + PublisherInt network file

Processing over multiple frames (lookBack and lookAhead options)

The above sections have described calculations within a frame, such as multiplying a vector by a constant. However, calculations over multiple frames (e.g. means) or differentiation requires processing *over* multiple frames. For example, to create a block to obtain the total of input values from two frames before to two frames after the current frame (total of five frames):

$$OUTPUT1(t) = INPUT1(t-2)+INPUT1(t-1)+INPUT1(t)+INPUT1(t+1)+INPUT1(t+2)| \quad (13.1)$$

Since `getInput` is maintained for each ID and `count` value of the input port, it can be calculated as:

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

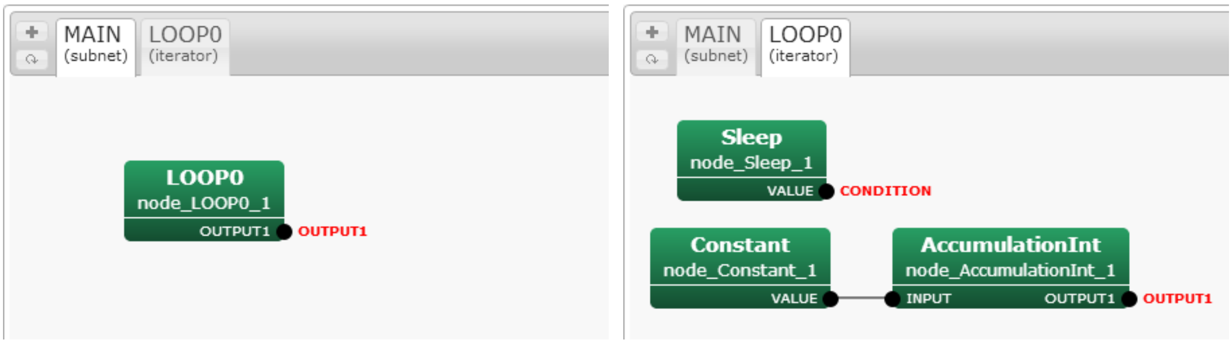
Here, the argument of `getInput` is expressed as "`count + i`", allowing processing from two frames before to two frames after the current frame. Errors during execution are due, first, to the `count` value having to be -2 and -1 in the first and second frames, respectively, resulting in a negative `count` value. This problem can be resolved simply by adding the following.

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

The second reason is that in FlowDesigner , unless particularly requested, only one frame before and one frame after the current frame are maintained during the process. Therefore, trying to see more than two frames away causes an error of undefined frames. This can be solved by describing the following in the constructor, enabling the buffer to maintain information for the desired number of frames.

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

The above is a command to take buffers from the two frames before and after the current frame. Set them adequately for your own purposes. This allows calculations for multiple frames. The following is the "`AccumulationInt`" node that takes an `int` type input and acquires a total number of frames, from `SUM_BACKWARD` frames before and `SUM_FORWARD` frames after the current frame. Since `lookAhead` and `lookBack` can be changed, errors will occur unless a sufficient number of frames are included in the buffer. An example of a network file is shown below.



(a) MAIN(subnet) Sheet (b) LOOP0(iterator) Sheet

Figure 13.5: 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, count + i);
        input1 = dereference_cast<int> (inputtmp);
        total += input1;
      }
    }
    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 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);
}
```

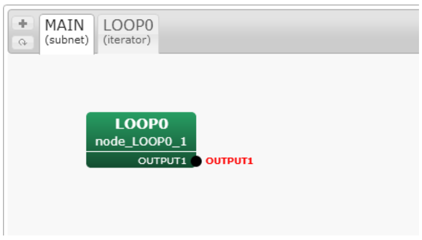
In this case, `getInput` is not read for every `count`, but once in three frames. In FlowDesigner , the former node is processed in accordance with the request from the `getInput`. That is, with a source like the above, the node request inputs once in three frames. Therefore, its former node is calculated once in three frames. 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"));
    in_order =dereference_cast<bool> (parameters.get("IN_ORDER"));
    inOrder = in_order;
  }

  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.
  }
};
```

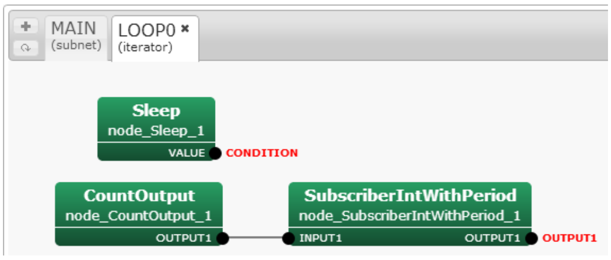
In the `SubscriberIntWithPeriod` node, `getInput` is performed over the period set in `PERIOD`, and its value is displayed. Next, create the `CountOutput` node to output the current `count` value without changing it.

```
#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, params)
  {
    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.



(a) MAIN(subnet) Sheet



(b) LOOP0(iterator) Sheet

Figure 13.6: SubscriberIntWithPeriod network file

First, set `IN_ORDER` of both `CountOutput` and `SubscriberIntWithPeriod` to "`false`" (These are default settings). Set `PERIOD` of `SubscriberIntWithPeriod` as 3, and execute the network file. The output of the console will be:

```
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]
...
```

Thus, since the latter `SubscriberIntWithPeriod` request inputs once in three frames, `CountOut` will be processed only once every three frames. If processing such as differentiation or count calculation is performed in `CountOut`, the process cannot be performed correctly. (To confirm this problem, implement a counter that counts every time `calculate` is called by the `CountOut` node. This will confirm that the counter does not work for all `count`.) The "`inOrder`" option solves this problem. Set `IN_ORDER` to `true` in the abovementioned network file. When executed, the following result is obtained:

```
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]
...
```

Thus, loop processing is properly executed for `CountOut` three times, in agreement with the requirement that it be performed once every three frames. To properly calculate the number of `count` times for all nodes, regardless of requirements, enter the following into the constructor:

```
inOrder = true;
```

Creating a node that takes a dynamic number of inputs (translateInput)

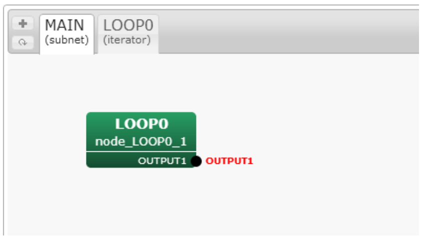
This section describes how to create a node that takes a dynamic number of inputs. The following example takes three `int` type inputs and calculates their total.

```
#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 = true;
  }

  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.
  }
};
```

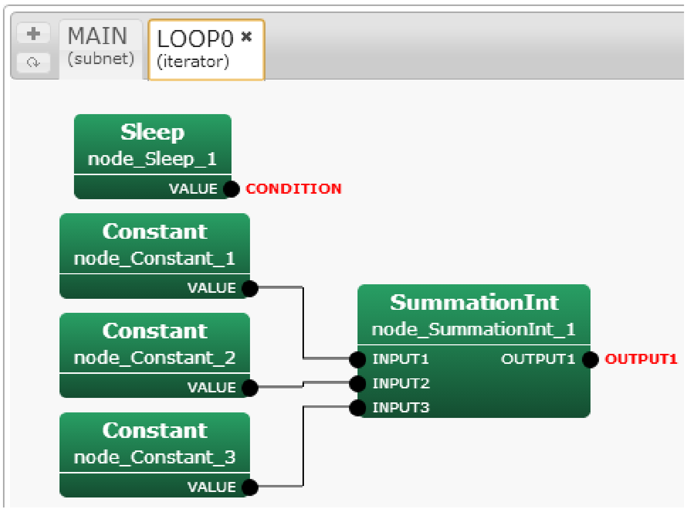
When the node is created, create a network file as follows.

In this case, 1+1+1, so 3 is the result. Currently, this node *has to* take three inputs. Now, suppose that you wish to sum *two* variables from this node, not all three. It thus appears that these calculations can be performed by opening one input port and giving integers to two ports, as shown in the following figure. However, when executing, an error occurred because the node requires all input ports to be connected.

In FlowDesigner , except in special cases, all information about the input ports read in `getInput` must be received, even if all the data are not used for processing inside the node. To realize the dynamic number of inputs, you can use a method called `translateInput`, which is in the member function of the `Node` class,



(a) MAIN(subnet) Sheet



(b) LOOP0(iterator) Sheet

Figure 13.7: SummationInt network file

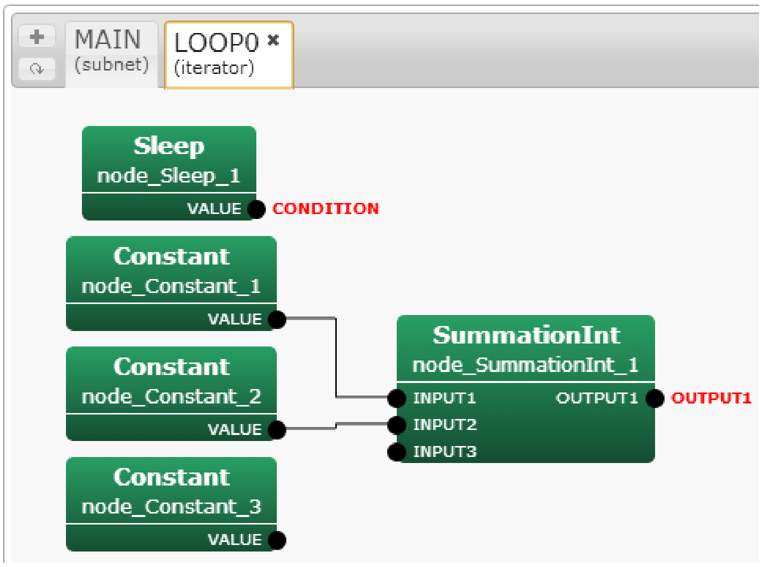


Figure 13.8: SummationInt network file (input port imperfection)

which is a parent class of `BufferedNode`. To create a node in which the number of input ports varies with the same example.

```
#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(-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 changes are as follows.

- The input port ID is initialized with -1 in the constructor.
  Thus, the ID will be kept at -1 unless the input port is connected.

- Added `translateInput`.
  This function detects which ports are connected. If an input port is connected, its ID will be acquired.

- Processing is performed in `calculate` for the input ports whose value is other than -1.
  This allows processing of examples in which the input is either opened or unopened.

Compiling this source and using it in FlowDesigner will show that the calculation has been performed properly, without depending on the number of inputs.

See Also

- Installation from source compilation
  See "Installation of HARK" in the HARK training session material

- How to make a basic node
  See "Creation of a node" in the HARK training session material

13.2 Improving the processing speed

Problem

How can I increase the computing speed of HARK?

Solution

The processing speed of HARK is dominated basically by the complexity of the nodes and their algorithms created by the user. As examples, the more you process eigenvalue expansion in LocalizeMUSIC , save of SaveFeatures or indications by `cout` and `cerr`, the longer the processing time required for one count. The simplest way of increasing processing speed is to construct simple nodes. (Algorithms have been improved for current HARK nodes to increase the speed of real time processing.) Two other methods can improve processing speed, although the improvements would be slight.

1) `Comment-in IN_ORDER_NODE_SPEEDUP`
   This function acts at the end of a node class. (Comment-in is described concretely in cc files such as LocalizeMUSIC)

2) Change the optimization option for compiling.
   Add this option while constructing the node. Optimization is performed with a stronger condition, in the order -O, -O1, -O2, and -O3, with processing speeds increasing accordingly.
   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, by analyzing the patterns compiled with options of -O, -O1, -O2, and -O3, and those for which `IN_ORDER_NODE_SPEEDUP` is further added. Thus, the processing times of eight patterns are compared. For comparison, use the algorithm for simple processing for each node, with processing times measured in 100 nodes connected in series.

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

Tables 13.1 and 13.2 show results without and with `IN_ORDER_NODE_SPEEDUP`, respectively. Computing times did not differ significantly, with processing speeds being only 3 percent higher with a combination of an optimization option and `IN_ORDER_NODE_SPEEDUP`.

See Also

None

Table 13.1: Elapsed Times without IN_ORDER_NODE_SPEEDUP

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**

Table 13.2: Elapsed Times with IN_ORDER_NODE_SPEEDUP

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**

13.3 Connecting HARK to the other systems

Problem

I understand that sound source localization and sound source separation can be performed in HARK. How can I transfer the information to other systems? To answer this question, we consider the followings:

- Connection between two different HARK systems
- Connection with other systems through a socket connection
- Connection with ROS(Robot Operating System : `http://www.ros.org/wiki/`)

Solution 1

: Connection between two different HARK systems

Users may wish to process several HARK network files as submodules of one network file. For example, when performing parallel computations 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 nodes must be treated. This section describes a method to treat one system described in the subnet sheet as one node.

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

2) **Export the created subnet sheet (Note 1)**
Make the created sheet active, and click
`"Networks" -> "Export Network"`.
Give it an appropriate name and save the created subnet sheet as a network file. Since the subnet sheet name will be maintained when it is imported later, we recommend giving it a proper subnet sheet name before exporting.

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

4) **Use it as a submodule**
Open the sheet that differ from the imported sheet. Right click on the new sheet and select the same name as that of the created subnet sheet from
`"New Node" -> "Subnet"`.
The module represents the subnet sheet itself with the same number of inputs and outputs as those of the imported subnet sheet.

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

Sub-modularization of this large system saves time into creating the same block construct and makes it easy to see the large-scale network file. The following are usage examples of this function.

A) **Performing the same processing multiple times**
When repeating the same processing or performing parallel computations, modular processing may allow the user to use the same node repeatedly in the main sheet.

B) **Always using the processing as a template**
If, for example, the same block configuration is always used for sound source localization, it would be easier to export the sound source localization processing once as a template and read it as one node every time, rather than to explicitly describe the processing of sound source localization every time.

(Note 1) Even without the exporting and importing functions, it is okay to copy and paste all nodes of a subnet sheet. Since, however, it is troublesome to copy and paste all the nodes of a large subnet sheet, it is recommended that the nodes be exported.

(Note 2) Modifications of submodules.

5-1) **Changing the numbers of inputs and outputs and of names**
The number of inputs/outputs and the names can be changed after importing. These changes require that you also change the main sheet to connect all inputs/outputs.

5-2) **Changing parameters in a submodule one by one**
Often, when using an imported submodule multiple times, the user will want to provide an argument, allowing the parameter set in the subnet sheet to be changed each time. If so, use `"subnet_param"`, a property parameter type. Its usage is described in the training section material.

Solution 2

:Connection with other systems through a socket connection
This chapter describes how to connect HARK systems with other systems by socket communication . The current HARK can be connected to Julius, a speech recognition engine, by socket communication. As a node for communication, the followings modules are in HARK by default.

1) SpeechRecognitionClient

2) SpeechRecognitionSMNClient

Using these modules, HARK clients can send feature vectors to Julius as messages through a socket, allowing Julius to process these messages. For details, see the above two sources. However, since these source codes are somewhat complicated, the following section describes simple methods of making a HARK node, both as a client and as a server.

Solution 2-1 :Connection with other systems through a socket connection (client)

In communicating with Julius, Julius continuously listens to messages from HARK as a server program. This example can be built by creating a simple server program and a HARK node. First create the server program `listener.c`, corresponding to a system other than HARK on 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[]){

  int i;
  int fd1, 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);
}
```

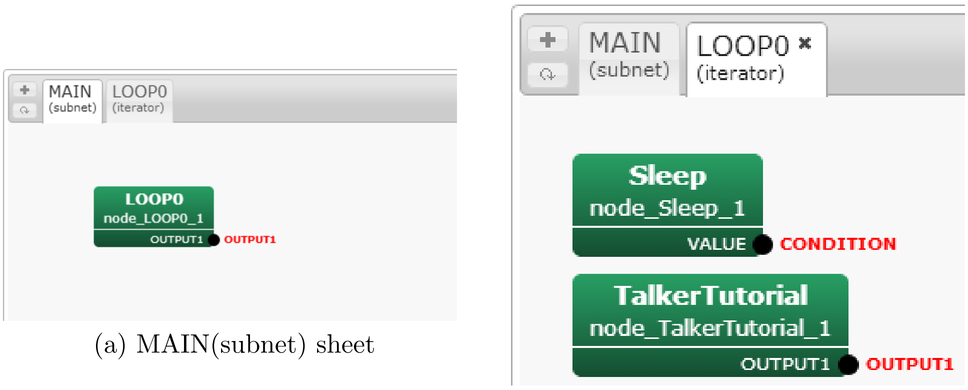
The content is a simple program for normal socket communication.
This program displays a message written in `fd2`. After cutting and pasting a source, compile it.

```
> gcc -o listener listener.c
```

Next, create a client node of HARK. 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("PORT"));
    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);
    }
```

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



(a) MAIN(subnet) sheet

(b) LOOP0(iterator) sheet

Figure 13.9: Network file : TalkerTutorial

To execute, first start the server program. Decide an appropriate port number (8765 here) and execute with the following command line.

```
> ./listener 8765
```

Next, launch 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 here)
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 or a host name to IP_ADDR (127.0.0.1 here)
```

To set an IP address, we have assumed that both the server and the client work with the same machine. Of course, it is possible to communicate with a remote machine.

```
Click "Execute" button
```

Messages from HARK are then transferred to the operating server, where they are indicated in each console as.

Server side (listener)

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

Client side (Flowdesigner: TalkerTutorial)

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

Solution 2-1 :Connection with other systems through a socket connection (server)

Next, create a server node that can receive data from a client program by socket communication. Similar to the solution above, a simple client program is prepared, followed by the creation of a server module. First create the server program `Talker.c`, 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];
  int ret;

  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);
}
```

Clearly, character strings read on a console are sent with the descriptor named `fd`. When the file is ready, compile it as:

```
> gcc -o talker talker.c
```

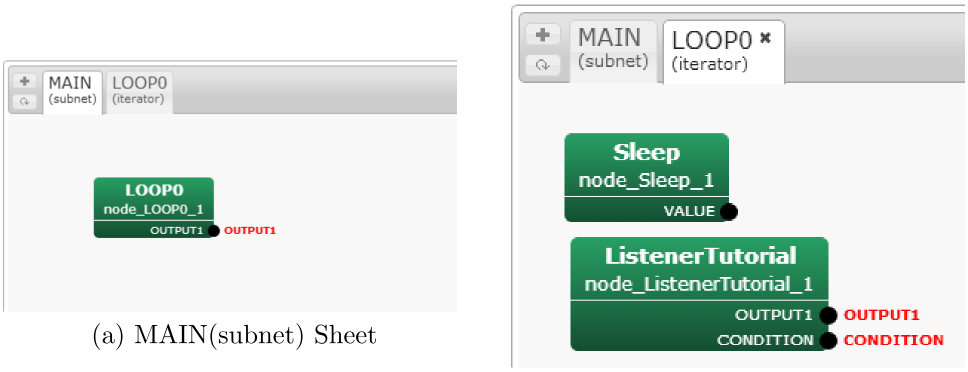
In building a server node in HARK, it is important to remember that the timing of messages sent by a client is unknown, making it necessary to create a server node so that a series of processes is performed every time a message is received. Therefore, it is necessary to create a in which the server itself becomes a trigger of new loop processing. For 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;
  }
};
```

This node differs from a normal node, in that it adds a port that outputs a new bool variable named `CONDITION`. This `CONDITION` port always returns `true` except for forced terminations and pauses in socket communication. This port can trigger loops of the network file of Flowdesigner. After building a network file, and cutting and pasting the above source, it can be compiled and installed. Flowdesigner is then started, and the following node created:



(a) MAIN(subnet) Sheet

(b) LOOP0(iterator) Sheet

Figure 13.10: Network file: ListenerTutorial

Here, the `CONDITION` output port became `CONDITION` of a loop of the network file. That is, the loop cycle of this network file was identical to the processing cycle of `ListenerTutorial`. The `ListenerTutorial` node suspends the processing until

```
ret = read(fd2, buf, 1024);
```

receives a new message. When a message is received, all `calculate` functions are processed and the processing of one `count` is completed. Making `CONDITION` of the network file the `CONDITION` port of the `ListenerTutorial` node enables the performance of a series of processes after one receipt of messages. To execute both, the server program (network file of Flowdesigner created above) is started.

```
Left-click the ListenerTutorial node >  Set the port number for PORT (8765 here)
Click the "Execute"
```

Make sure to set `CONDITION`. Start the client program next. Start the new console and move to a directory that stores `talker.c`, as compiled above. Execute it using the command line.

```
> ./talker 127.0.0.1 8765
```

Assuming that the server and client work with the same machine, we have used the IP address `127.0.0.1`, although it is possible to communicate with a remote machine. Now, enter some characters into the console of `talker`. Messages from the client console will be communicated to the server node of operating Flowdesigner. They are indicated in each console as (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, a HARK node itself can act as a server and receive a messages from other systems.

Solution 3

: Connection with ROS (Robot Operating System)

If your system was created in ROS, the open source platform for robots developed by Willow Garage, you can create a communication between HARK and ROS using the instructions in this section.

Since a sufficient documents for ROS are available on the web, we assume here that (1) an ROS system has already been installed and, (2) the user knows the basic concepts of ROS, including publish to topic and subscribe from topic; i.e., we assume that the user has completed the Beginner Level of the ROS tutorial. Since there are two connection methods, each is described here. Moreover, Python is used for implementation of the ROS node.

(1) Use standard output This 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 this 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 python script with the following code.

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

Outputs of this network can be processed in python script with the code.

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

The information acquired from HARK can be published to an appropriate topic.

(2) Use of socket communication

Another method is to connect ROS and HARK via socket communication.

It will therefore be necessary to create a code of socket communication on the ROS side and a node on the HARK side. Despite difficulties, the entire configuration is clear for users. For creation of nodes for HARK, see the tutorial in the HARK documents. 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

This is a communication method utilizing the ROS system. If a node to publish a message for a topic of ROS is created, localization results and separated sounds can be sent directly to ROS.

See Also

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

13.4 Controlling a motor

Problem

Solution

Since HARK itself is software for audition processing, a module describing the control of a motor is not included. However, when performing comparatively simple control, a module can be designed to control a motor. This section describes how a method of implementing this function in the client part of HARK as a module by dividing control of a motor into applications of server and client. The following is an example of the main processing of the client module.

```
function calculat
1. if count = 0 then
2.       set_IP_address_and_PORT_number
3.       connect_to_server
4. endif
5. obtain_input
6. generate_control_input
7. send_control_input
8. check_motor_state
```

```
function destructor
1. disconnect_to_server
```

The first calculate function performs processing related to TCP/IP connection once when it is first activated and repeats the processing for other cases. To access TCP/IP, the first activation of this function sets the assigned IP address port and connects to the server application (the second and third lines). When it is first connected to the server application, it repeats processing to generate a control command from an input. After an input is received, a command for motor control is created based on the data. This command is sent to the server application, which confirms whether the command was sent properly. The second destructor function indicates a destructor of a module and performs processing to cut off connections with the server. 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 its description is therefore omitted here.

Discussion

Since Flowdesigner is data flow-oriented, the presence of even one node that requires a long time for processing, makes real-time processing difficult in some cases. Therefore, processing on Flowdesigner consists of generating a command to control a motor, with the load reduced by operating the motor in other server applications. Moreover, dividing into a server and a client enables another motor to be controlled simply by changing the interface.

See Also

For motor control, see “Creation of node” in 12.1. For the combined control of a more complicated motor or other sensors, see “Control of robot” in 12.2.

Chapter 14

Appendix

14.1 Introduction

This chapter introduces the network sample with frequent modules. Sample networks are available from HARK wiki. 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. The sample networks are categorized for each function. The description is summarized simple as possible and therefore it is for training of HARK modules.

14.1.1 Category of sample network

All categories of sample networks are shown in Table 14.1. 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.

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, when you want to use a script file named demo.n, execute the following files:

For Ubuntu

The script name is "demo.sh."

For Windows

The batchfile is "demo.bat."

Character strings before "." are commonized. However, demo.sh may include the setting items that depend on operating environments (i.e. IP address) and therefore unexpected result may occur when executing without confirmation. The correct setting method is described in description of each sample network.

Table 14.1: 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

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.

- **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 DisplayLocalization and saved in a file with SaveSourceLocation is 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 HARK is already installed. In order to execute online localization processing, AD/DA for the multi-channel recording that HARK supports is required.

- **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 GHDSS and PostFilter or GHDSS and 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.

- **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 MSLSExtraction and 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 MFM category are used. This is a sample in particular for usage of MFMGeneration and SpeechRecognitionClient . A sample in which off-line sound source localization processing is performed to recorded sounds for eight channels is available.

14.1.2 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 part of the operation example indicates an input from the user, and the second line 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

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

Figure 14.1: Execution example of sample network start-up script

.

environments. The 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 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.

14.2 Sound recording network sample

When you record signals using a device, you can use sample network files included in the RecordLin and RecordWin folders. We explain samples for Ubuntu in Section 14.2.1 and for Windows in Section 14.2.2, respectively.

14.2.1 Ubuntu

Four kinds of sound recording network samples are available as shown in Table 14.2. The left column indicates network files and the right column indicates processing contents.

Table 14.2: Sound recording network list.

Network file name	Processing content
demoALSA.n	ALSA sound recording
demoWS_8ch.n	Sound recording for eight channels with RASP
demoRASP24_8ch.n	Sound recording for eight channels with RASP24

For ALSA devices, you can use a shell script demoALSA.sh for recording various number of frames, number of channels, and device names. For example, if you want to record 200 frames from an 2-ch audio device whose name is plughw:1,0, you can use this script as shown below:

```
> ./demoALSA.sh 200 2 plughw:1,0
 UINodeRepository::Scan()
 Scanning def /usr/lib/flowdesigner/toolbox
 done loading def files
 loading XML document from memory
 done!
 Building network  :MAIN

 <Bool 1 >
```

Figure 14.2: Execution example of demoALSA.sh.

Note that the difference between these networks is only the parameters of AudioStreamFromMic module. The DEVICE parameter (plughw:1,0, 127.0.0.1, etc.) changes according to an environment. Therefore, an user has to set correct value.

A sample network consists of two sub-networks (MAIN, MAIN_LOOP). MAIN (subnet) and MAIN_LOOP (iterator) have one and six modules, separately. Figure 14.3 and 14.4 shows MAIN (subnet) and MAIN_LOOP (iterator). It is simple network configuration in which the audio waveforms collected in the AudioStreamFromMic module are selected in ChannelSelector , and written in files in SaveWavePCM . The SaveWavePCM module connected to OUTPUT1 saves a file with eight channel signal. The other SaveWavePCM module saves eight files with each channel signal.

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.

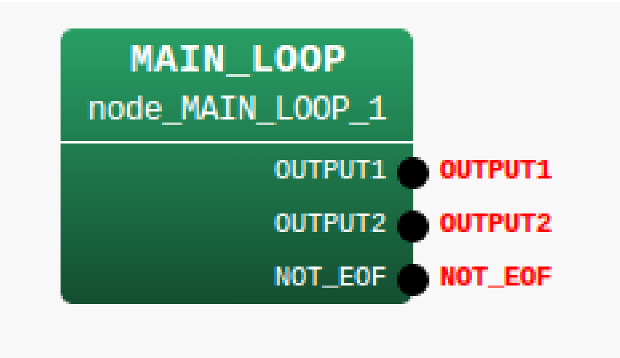


Figure 14.3: MAIN (subnet)

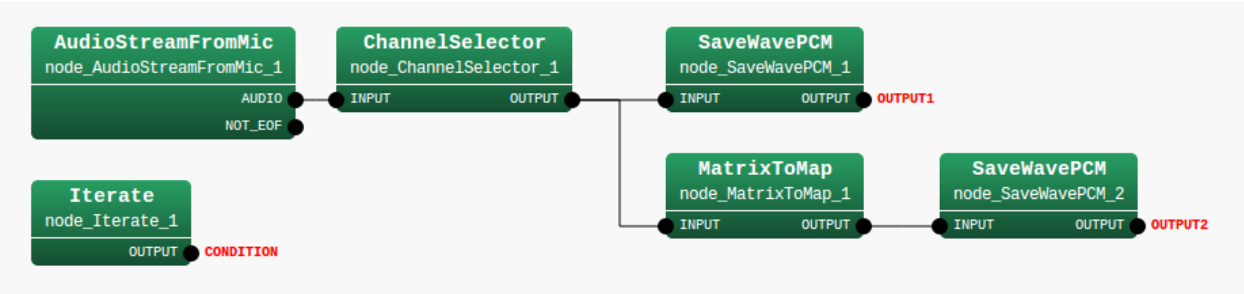


Figure 14.4: MAIN_LOOP (iterator)

There are five properties for the MAIN_LOOP node in MAIN (subnet). Table 14.3 shows a list of them. SAMPLING_RATE and GET_FRAMES are important. Each parameter value is set according to the values in the table. Set value of GET_FRAMES is `int` :ARG1 here. This means the monobasic argument of demoALSA_2ch.n casted to the integral type and substituted. The number of sound recording frames is given to this argument in demoALSA_2ch.n in demoALSA_2ch.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.

$$(LENGTH + (GET\_FRAMES - 1) * ADVANCE)/SAMPLING\_RATE \tag{14.1}$$

Table 14.3: Parameter list of **MAIN_LOOP**

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
GET_FRAMES	`subnet_param`	`int` :ARG1		Number of frames for sound recording
DOWHILE	`bool`			Blank

Stereo recording with ALSA-based devices

Execute demoALSA_2ch.sh in the Record directory. After the execution, one stereo file rec_all_0.wav and two monaural files rec_each_0.wav, rec_each_1.wav are generated. When replaying these files, the recorded content can be confirmed.

Some computer rarely do not accept stereo recording so an error may occur. In the case that the 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 nodes are included in this sample. There is one node in MAIN (subnet) and are four nodes in MAIN_LOOP (iterator). It is simple network configuration in which the audio waveforms collected in the AudioStreamFromMic module are written in a file in SaveWavePCM . Data formats between nodes are uniformed through MatrixToMap . Iterate is a node 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 CHANNEL_COUNT property of the AudioStreamFromMic node. It is 1 for monaural recording and 2 for stereo recording.

Recording cannot be performed well, perform the following checks.

1. 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.

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

3. 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.

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.

Parameters of AudioStreamFromMic are shown in Table 14.4. In this node, designate a sound recording device. Designate a sound recording device in this node. In this demo, the number of sound recording channels is 2ch 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 14.4: Parameter list of AudioStreamFromMic

Parameter name	type	Set value	Unit	Description
LENGTH	`subnet_param`	LENGTH	[pt]	FFT length
ADVANCE	`subnet_param`	ADVANCE	[pt]	Shift length
CHANNEL_COUNT	`int`	2	[ch]	Number of sound recording channel
SAMPLING_RATE	`subnet_param`	16000	[Hz]	Sampling frequency
DEVICETYPE	`string`	ALSA		Device type
DEVICE	`string`	plughw:0,0		Device name

8ch sound recording with radio RASP

Execute demoWS_8ch.sh in a Record directory, after setting several items. Although 127.0.0.1 is designated as an IP address of the radio RASP in demoWS_8ch.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 RASP for the method to execute ws_fpaa_config. After the execution, one file rec_all_0.wav and eight monaural files rec_each_0.wav, rec_each_1.wav, ..., rec_each_7.wav are generated. When replaying these files, the recorded content can be confirmed.

When recording cannot be performed well, perform the following checks.

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.

Parameters of AudioStreamFromMic are shown in Table 14.5. Designate a sound recording device in this node. 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.

Note that when WS is used, CHANNEL_COUNT must be 16. Therefore, when a user want to record eight channel sound with WS, use ChannelSelector to select eight channels from 16 channels. In this sample, from 0 to 7 channels are selected.

Table 14.5: Parameter list of AudioStreamFromMic

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

8ch sound recording with RASP24

Execute demoRASP24_8ch.n in the Record directory. After the execution, one file rec_all_0.wav and eight monaural files rec_each_0.wav, rec_each_1.wav, ..., rec_each_7.wav are generated. When replaying these files, the recorded content can be confirmed.

When recording cannot be performed well, perform the following checks.

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 RASP24. Check the network connection with ping.

3. A correct IP address of RASP24 is set to AudioStreamFromMic .

4. 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.

Parameters of AudioStreamFromMic are shown in Table 14.6. RASP24 can record sound with 16 bit or 32 bit. When perform 16 bit and 32 bit recording, use RASP24-16 and RASP24-32, separately. Designate a sound recording device in this node. Designate a multi-channel sound recording device. Set 16ch to the number of sound recording channel and IP address in DEVICE. In order to execute the sample, it is necessary to change the address to that of the actual radio RASP24.

Note that a parameter GAIN is added when RASP24 is used as shown in Figure 14.5.

Table 14.6: Parameter list of AudioStreamFromMic

Parameter name	type	Set value	Unit	Description
LENGTH	`subnet_param`	LENGTH	[pt]	FFT length
ADVANCE	`subnet_param`	ADVANCE	[pt]	Shift length
CHANNEL_COUNT	`int`	16	[ch]	Number of sound recording channel
SAMPLING_RATE	`subnet_param`	16000	[Hz]	Sampling frequency
DEVICETYPE	`string`	RASP24-16		Device type
GAIN	`string`	0dB		GAIN
DEVICE	`string`	127.0.0.1		Device name

When a user set GAIN parameter, be careful about clipping.

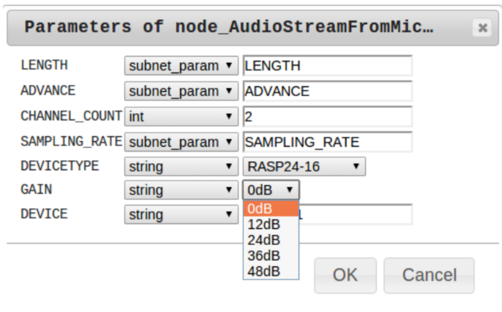


Figure 14.5: Additional parameter for RASP24

14.2.2 Windows

4ch Sound Recording with DS-based devices

Execute demoDS_4ch.sh in the RecordWin directory. After the execution, one 4ch file rec_all_0.wav and two monaural files rec_each_0.wav, rec_each_1.wav are generated. When replaying these files, the recorded content can be confirmed.

Recording cannot be performed well, perform the following checks.

1. 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.

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

3. 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.

4. When more than two audio interfaces are connected, remove audio interfaces after the second ones before recording the sound.

5. When you use Kinect, you must install a drive in advance.

Parameters of AudioStreamFromMic are shown in Table 14.7. In this node, designate a sound recording device. Designate a sound recording device in this node. In this demo, the number of sound recording channels is 4ch so that a lot of users can check the operation, and DS is designated for DEVICETYPE and Kinect is designated for DEVICE.

Table 14.7: Parameter list of AudioStreamFromMic

Parameter name	type	Set value	Unit	Description
LENGTH	subnet_param	LENGTH	[pt]	FFT length
ADVANCE	subnet_param	ADVANCE	[pt]	Shift length
CHANNEL_COUNT	int	4	[ch]	Number of sound recording channel
SAMPLING_RATE	subnet_param	16000	[Hz]	Sampling frequency
DEVICETYPE	string	DS		Device type
DEVICE	string	Kinect		Device name

8ch sound recording with RASP24

Execute demoRASP24_8ch.n in the RecordLin directory. After the execution, one file rec_all_0.wav and eight monaural files rec_each_0.wav, rec_each_1.wav, ..., rec_each_7.wav are generated. When replaying these files, the recorded content can be confirmed.

When recording cannot be performed well, perform the following checks.

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 RASP24. Check the network connection with ping.
3. A correct IP address of RASP24 is set to AudioStreamFromMic .
4. 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.

Parameters of AudioStreamFromMic are shown in Table 14.6.

14.3 Network sample of sound source localization

We provide two network sample files for sound source localization. You can execute all network files using `demo.sh`. Table 14.8 show the network file name, how to run it, and the description.

Table 14.8: Sound source localization network list.

Network file name	Execution	Content
demoOffline.n	demo.sh offline	Sound source localization of 4ch audio file
demoOnline.n	demo.sh online	Online sound source localization using Kinect

14.3.1 Offline sound source localization

Execution

Run the offline localization in Localization directory. This script displays the sound source localization result of three 4ch audio file that inclues simultaneous speech of two speakers from -30 and 30 degrees (`Multispeech.wav`). You can find the file in data directory.

You can run the script by

```
> ./demo.sh offline
```

Then, you will see the output like Figure 14.6 in the terminal, and a graph of sound source localization.

```
UINodeRepository::Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network  :MAIN
TF was loaded by libharkio2.
1 heights, 72 directions, 1 ranges, 7 microphones, 512 points
Source 0 is created.
 (skipped)
Source 5 is removed.
UINodeRepository::Scan()
 (skipped)
```

Figure 14.6: Result of offline localization

Checking the results

After you ran the script, you will find two text files: Localization.txt and log.txt. If you cannot find them, you failed the execution. Check the following things:

1. Check if MultiSpeech.wav is in the ../data directory. These files are real recorded sound by Kinect, which is supported by HARK. You cannot run the network since this is the input file.

2. Check if the kinect_loc.zip file is in the ../config directory. They are the transfer function file for localiztion. See LocalizeMUSIC for details.

Localization.txt contains sound source localization results generated by SaveSourceLocation in a text format. They contain the frame number and sound source direction for each frame. You can load them using LoadSourceLocation . See HARK document for the format of the file.

log.txt contains the MUSIC spectrum generated by LocalizeMUSIC with DEBUG property `true`. MUSIC spectrum means a kind of confidence of the sound existence, calculated for each time and direction. The sound exists if the MUSIC spectrum value is high. See LocalizeMUSIC in HARK document for details. We also provide the visualization program of MUSIC spectrum. Run the following commands:

```
> python plotMusicSpec.py log.txt
```

If you fail, your system may not have necessary python package. Run the following command:

For Ubuntu

```
sudo apt-get install python python-matplotlib
```

This result can be used for tuning the THRESH property of SourceTracker .

Sample network description

This sample has nine nodes. Three nodes are in MAIN (subnet) and six nodes are in MAIN_LOOP (iterator). MAIN (subnet) and MAIN_LOOP (iterator) are shown in Figures 14.7 and 14.8. AudioStreamFromWave loads the waveform, MultiFFT transforms it to spectrum, LocalizeMUSIC localizes the sound, SourceTracker tracks the localization result, then, DisplayLocalization shows them and SaveSourceLocation stores as a file.

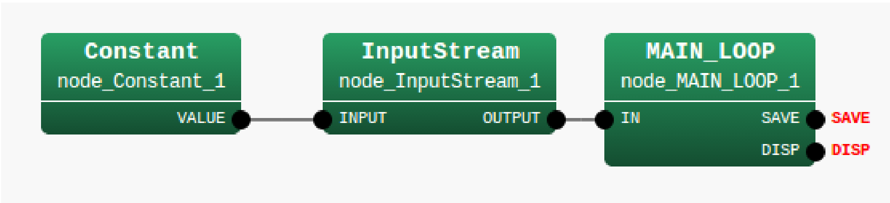


Figure 14.7: MAIN (subnet)

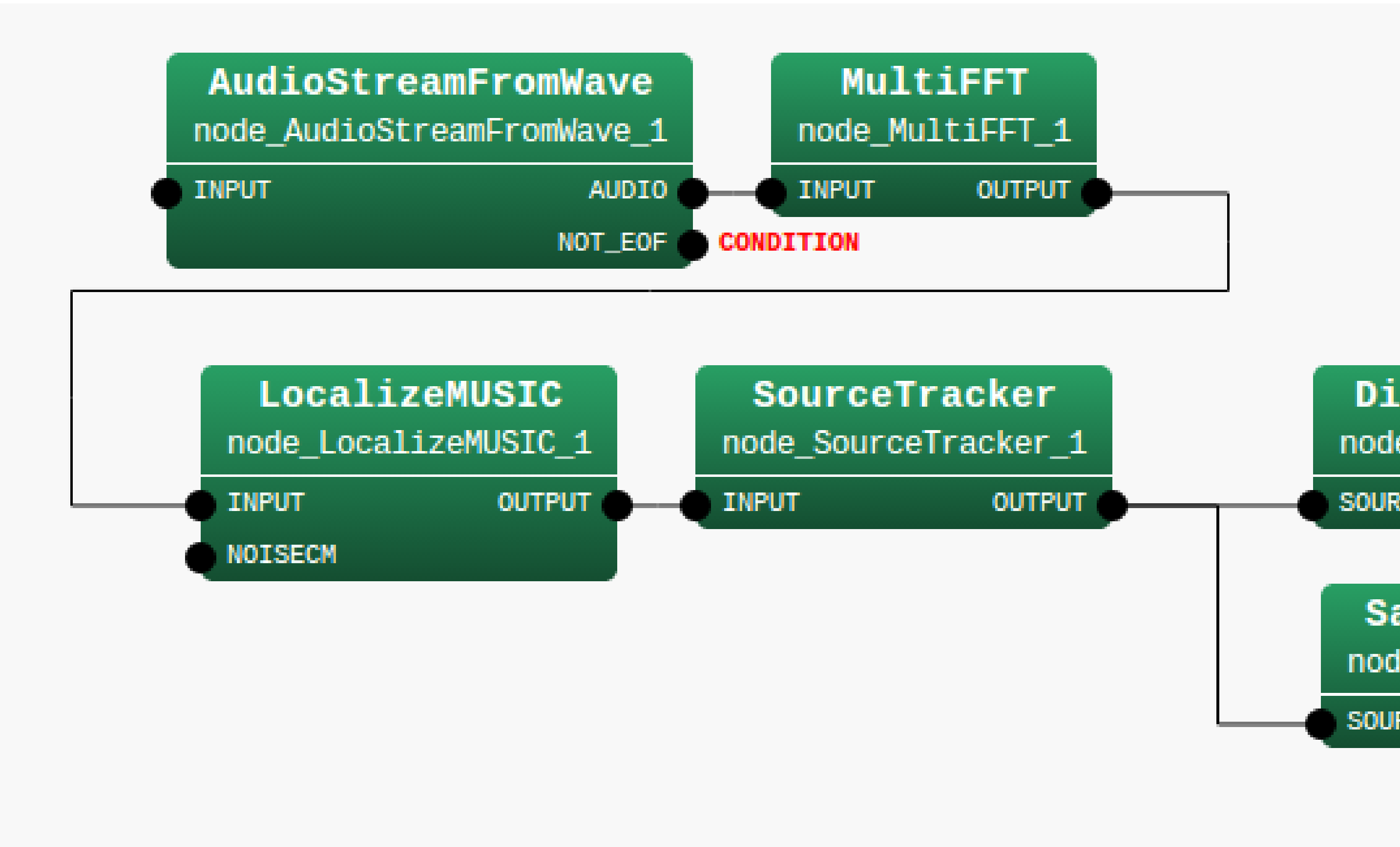


Figure 14.8: MAIN_LOOP (iterator)

Table 14.9 summarizes the main parameters. The most important parameter is A_MATRIX, which specifies a file name of transfer function for localization. If you use a microphone array we support, you can download them from HARK web page. However, if you want use your own microphone array, you need to make it by harktool.

14.3.2 Online sound source localization

Execution

First, connect your Kinect to a USB port of your computer. Then, run the following command:

Table 14.9: Parameter list

Node name	Parameter name	Type	Value
MAIN_LOOP	LENGTH	int	512
	ADVANCE	int	160
	SAMPLING_RATE	int	16000
	A_MATRIX	int	ARG2
	FILENAME	subnet_param	ARG3
	DOWHILE	bool	(empty)
LocalizeMUSIC	NUM_CHANNELS	int	4
	LENGTH	subnet_param	LENGTH
	SAMPLING_RATE	subnet_param	SAMPRING_RATE
	A_MATRIX	subnet_param	A_MATRIX
	PERIOD	int	50
	NUM_SOURCE	int	1
	MIN_DEG	int	-90
	MAX_DEG	int	90
	LOWER_BOUND_FREQUENCY	int	300
	HIGHER_BOUND_FREQUENCY	int	2700
	DEBUG	bool	false

```
> cat /proc/asound/cards
 0 [AudioPCI       ]: ENS1371 - Ensoniq AudioPCI
                      Ensoniq AudioPCI ENS1371 at 0x2080, irq 16
 1 [Audio          ]: USB-Audio - Kinect for Windows USB Audio
                      Microsoft Kinect for Windows USB Audio at usb-0000:02:03.0-1, high speed
```

If you see the word "Kinect", the OS successfully recognized it. In this case, its device name is `plughw:1` because the number shown in the left side of "Kinect" is one. If the number is not `plughw:1`, open `demo.sh` and edit DEVICE.

Then, run the script

```
> ./demo.sh online
```

You will see the result like Figure 14.9 in your terminal, and visualized sound locations.

```
UINodeRepository::Scan()
Scanning def /usr/lib/flowdesigner/toolbox
done loading def files
loading XML document from memory
done!
Building network  :MAIN
TF was loaded by libharkio2.
1 heights, 72 directions, 1 ranges, 7 microphones, 512 points
Source 0 is created.
Source 0 is removed.
  (skipped)
```

Figure 14.9: Execution example of online sound source localization

Checking the result

If you have problem on localization, check the same things as offline localization. You can also refer to the recipe: Localization failed.

Sample network description

Seven nodes are included in this sample. There is one node in MAIN (subnet) and are six nodes in MAIN_LOOP (iterator). MAIN (subnet) and MAIN_LOOP (iterator) are shown in Figures ?? and 14.11. AudioStreamFromMic records the sound. SaveWavePCM stores the sound. Simultaneously, MultiFFT transforms it to spectral representation, LocalizeMUSIC localizes it for each frame, SourceTracker tracks using temporal connectivity, and DisplayLocalization displays it.

Table 14.10 summarizes the main parameters.

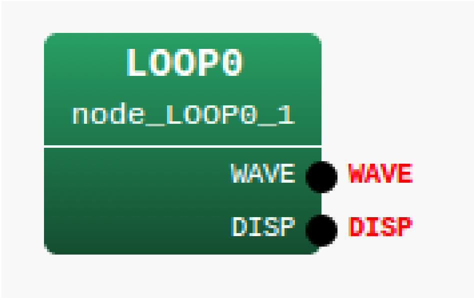


Figure 14.10: MAIN (subnet)

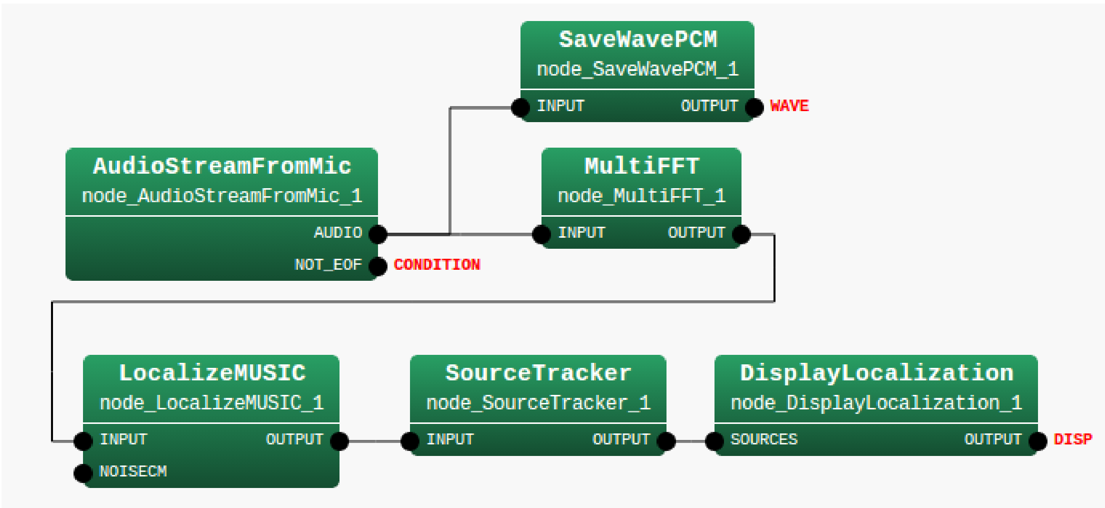


Figure 14.11: MAIN_LOOP (iterator)

Table 14.10: Parameter list

Node name	Parameter name	Type	Value
MAIN_LOOP	LENGTH	`int`	512
	ADVANCE	`int`	160
	SAMPLING_RATE	`int`	16000
	A_MATRIX	`string`	ARG1
	DOWHILE	`bool`	(empty)
LocalizeMUSIC	NUM_CHANNELS	`int`	4
	LENGTH	`subnet_param`	LENGTH
	SAMPLING_RATE	`subnet_param`	SAMPRING_RATE
	A_MATRIX	`subnet_param`	A_MATRIX
	PERIOD	`int`	50
	NUM_SOURCE	`int`	1
	MIN_DEG	`int`	-90
	MAX_DEG	`int`	90
	LOWER_BOUND_FREQUENCY	`int`	300
	HIGHER_BOUND_FREQUENCY	`int`	2700
	DEBUG	`bool`	`false`

14.4 Network sample of sound source separation

There are four kinds of network samples for sound source separation as shown in Table 14.11. The left column indicates network files, the middle column indicates shell script files to operate the networks and the right column indicates processing contents.

You can run these samples using `demo.sh` as follows.

```
> ./demo.sh offline        # Offline processing without HRLE
> ./demo.sh offline HRLE   # Offline processing with HRLE
> ./demo.sh online         # Online processing without HRLE
> ./demo.sh online HRLE    # Online processing with HRLE
```

Table 14.11: Sound source separation network list.

Network file name	Network execution script	Processing content
demoOffline.n demoOfflineHRLE.n	demoOffline.sh demoOfflineHRLE.sh	Sound source separation of 4ch audio Processing for which noise estimation by HRLE as a postprocessing
demoOnline.n demoOnlineHRLE.n	demoOnline.sh demoOnlineHRLE.sh	Sound source separation of captured audio Processing for which noise estimation by HRLE as a postprocessing

14.4.1 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 can confirm while executing sound source separation.

Run demo.sh in Separation directory with a command line argument "offline". After you run, you will see the separated sounds in `sep_files/`. The file names are `offline_%d.wav`.

If you failed run the sample, check the following things.

1. Check if the transfer function files kinect_loc.zip, kinect_sep.zip files are in the ../config directory.

2. Check if MultiSpeech.wav is in the ../data directory.

Twelve nodes are included in this sample. There are three nodes in MAIN (subnet) and are nine nodes in MAIN_LOOP (iterator). MAIN (subnet) includs Constant , InputStream , and MAIN_LOOP (iterator). MAIN_LOOP (iterator) is shown in Fig. 14.12 The audio waveforms read from the files in the AudioStreamFromWave node are analyzed in MultiFFT , separated in GHDSS , synthesized in Synthesize and the audio waveforms are saved in SaveWavePCM .

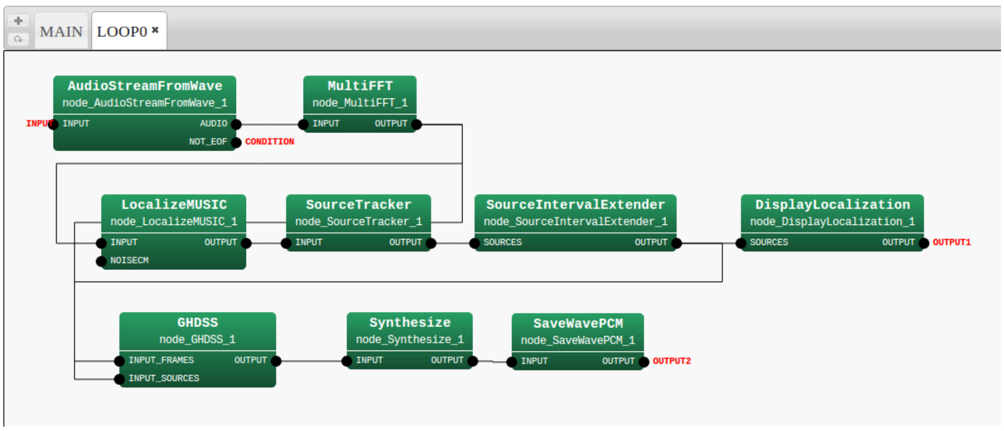


Figure 14.12: Sound source separation without HRLE

An important parameter is TF_CONJ_FILENAME. Use the file created in harktool3 from impulse responses of a kinect.

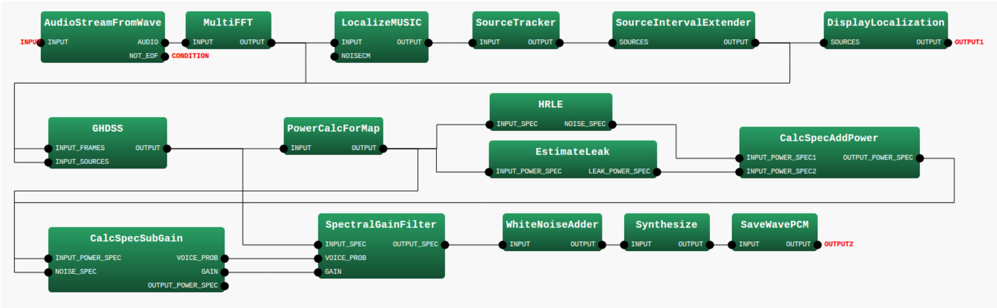


Figure 14.13: MAIN_LOOP (iterator)

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

A sample network file that separates sound by GHDSS node and post process for speech enhancement using HRLE is introduced here.

Run the demo.sh with commandline arguments `offline` and `HRLE`. THen, you will find the separated files in `sep_files/` directory. The file name is `offlineHRLE_%d.wav`

Figure 14.13 shows sample network of demoOfflineHRLE.n, which is off-line sound source separation with postprocessing by HRLE . The audio waveforms read from the files in the AudioStreamFromWave node are analyzed in MultiFFT , separated in GHDSS , postprocessed, synthesized in Synthesize and the audio waveforms are saved in SaveWavePCM . The postprocessing is realized by combinations of HRLE , Estimate-Leak , CalcSpecAddPower , CalcSpecSubGain and 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.

14.4.3 Online sound source separation (with/without postprocessing by HRLE)

Samples of online sound source separation are introduced here. You need Kinect for these samples.

If you want to run with HRLE post processing, run demo.sh with command line arguments `online` and `HRLE`. If you want to run without HRLE , run demo.sh with a command line argument `online`.

If you successfully finished running, you will find the graphical localization results and the separated files in `sep_files/` directory. The file names are `online_%d.wav` and `onlineHRLE_%d.wav`. You can stop by pressing the Control key and 'c' key at the same time.

If you failed running, check the network or device using chapter 3.

14.5 Network samples of Feature extraction

14.5.1 Introduction

HARK provides static feature extraction modules such as MSLSExtraction and MFCCExtraction . HARK also provides following additional feature extraction modules and preprocessing modules:

1. Dynamic feature (Delta term): The temporal change of the static features. Notated as ΔMSLS in Tab. 14.12. Calculated by Delta .
2. Power: The power of the input signal. Notated as Power in Tab. 14.12. Calculated by PowerCalcForMap .
3. Delta power: The temporal change of power. Notated as ΔPower in Tab. 14.12. Calculated by Delta
4. Preprocessing: Emphasizing of the high frequency range using PreEmphasis or mean normalization. Notated as Preprocessing in Tab. 14.12.

We provide six sample networks of acoustic feature extraction as shown in Tab. 14.12. From the left, the three columns of the table denotes: the network file name, the feature description, and the feature file name generated by the network file. You can run the samples by executing `demo.sh`. For example, if you want to execute `demo1.n`, run

```
> ./demo.sh 1
```

The samples calculate the features using the MSLS of 13 dimensions in offline. If you want to calculate features in online, replace AudioStreamFromWave with AudioStreamFromMic . If you want to use MFCC instead of MSLS, replace MSLSExtraction with MFCCExtraction . If you want to change the dimension of the MSLS, change the property. See HARK document for details.

Table 14.12: Sample files of feature extraction

Network file name	Feature						Generated file
	MSLS 13 dim	Δ MSLS 13 dim	Power 1 dim	ΔPower 1 dim	Preprocessing no dim	Corresponding section	
demo1.n	Yes					14.5.2	MFBANK13_0.spec
demo2.n	Yes	Yes				14.5.3	MFBANK26_0.spec
demo3.n	Yes		Yes			14.5.4	MFBANK14_0.spec
demo4.n	Yes	Yes	Yes	Yes		14.5.5	MFBANK28_0.spec
demo5.n	Yes	Yes		Yes		14.5.6	MFBANK27_0.spec
demo6.n	Yes	Yes		Yes	Yes	14.5.7	MFBANK27p_0.spec

14.5.2 MSLS

An execution example is shown in Figure 14.14. After the execution, a file named `MFBANK13_0.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 if the f101b001.wav files are in the data directory.

```
> ./demo.sh 1
 MSLS
 UINodeRepository::Scan()
 Scanning def /usr/lib/flowdesigner/toolbox
 done loading def files
 loading XML document from memory
 done!
 Building network  :MAIN
```

Figure 14.14: Execution example

Twelve modules are included in this sample. There are three modules in MAIN_LOOP (iterator) and nine modules in MAIN (subnet). MAIN (subnet) and MAIN_LOOP (iterator) are shown in Figure 14.15 and Figure 14.16. As an outline of the processing, it is simple network configuration in which acoustic features are calculated in MSLSExtraction with the audio waveforms collected in the AudioStreamFromWave module and are written in SaveFeatures . Since MSLSExtraction requires the outputs of the mel-scale filter bank and power spectra for calculation of MSLS, the collected audio waveforms are analyzed by MultiFFT and their data type are converted by MatrixToMap and PowerCalcForMap , and then processing to obtain outputs of the mel-scale filter bank is performed by MelFilterBank . MSLSExtraction reserves 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 FBANK_COUNT property of MSLSExtraction as 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 FeatureRemover to delete it. SaveFeatures saves the input FEATURE. The localization result from the front generated by ConstantLocalization is gave to SOURCES.

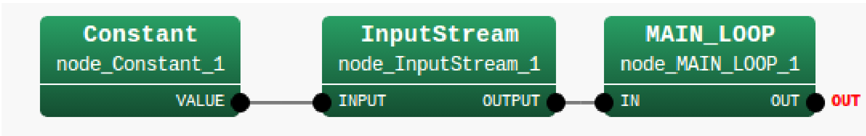


Figure 14.15: MAIN (subnet)

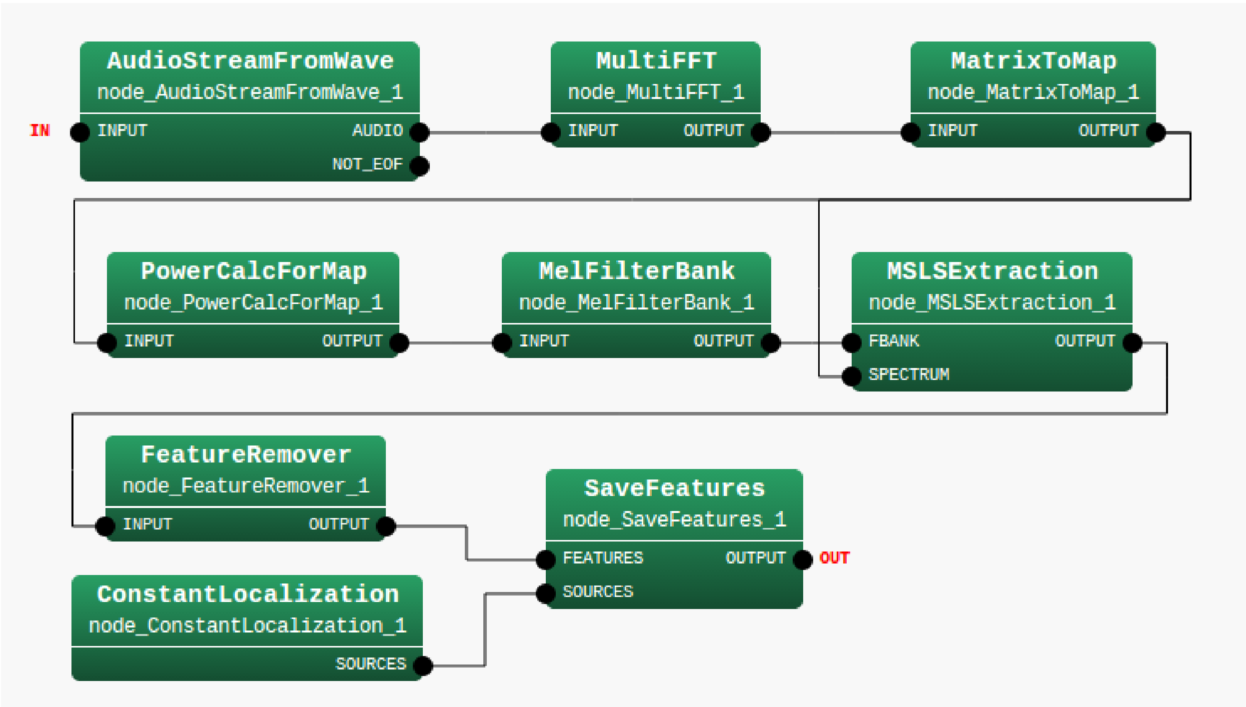


Figure 14.16: MAIN_LOOP (iterator)

Table 14.13 summarizes the main modules. The most important module is MSLSExtraction .

Table 14.13: Parameter list

Node name	Parameter name	Type	Value
MAIN_LOOP	LENGTH	subnet_param	int :ARG2
	ADVANCE	subnet_param	int :ARG3
	SAMPLING_RATE	subnet_param	int :ARG4
	FBANK_COUNT	subnet_param	int :ARG5
	DOWHILE	bool	(empty)
MSLSExtraction	FBANK_COUNT	subnet_param	FBANK_COUNT
	NORMALIZE_MODE	string	Cepstral
	USE_POWER	bool	false

14.5.3 MSLS + ΔMSLS

An execution example is shown in Figure 14.17. After the execution, a file named `MFBANK26_0.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 if the f101b001.wav files are in the data directory.

```
> ./demo.sh 2
 UINodeRepository::Scan()
 Scanning def /usr/lib/flowdesigner/toolbox
 done loading def files
 loading XML document from memory
 done!
 Building network :MAIN
```

Figure 14.17: Execution example

Twelve modules are included in this sample. There are three modules in MAIN_LOOP (iterator) and nine modules in MAIN (subnet). MAIN (subnet) and MAIN_LOOP (iterator) are shown in 14.18 and 14.19. As an outline of the processing, it is simple network configuration in which acoustic features are calculated in MSLSExtraction with the audio waveforms collected in the AudioStreamFromWave module and are written in SaveFeatures . Since MSLSExtraction requires the outputs of the mel-scale filter bank and power spectra for calculation of MSLS, the collected audio waveforms are analyzed by MultiFFT and their data type are converted by MatrixToMap and PowerCalcForMap , and then processing to obtain outputs of the mel-scale filter bank is performed by MelFilterBank . MSLSExtraction reserves 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 FBANK_COUNT property of MSLSExtraction as a feature. zero is in the storing region for theδ MSLS coefficient. The δ MSLS coefficient is calculated and stored with Delta . SaveFeatures saves the input FEATURE. The localization result from the front generated by ConstantLocalization is gave to SOURCES.

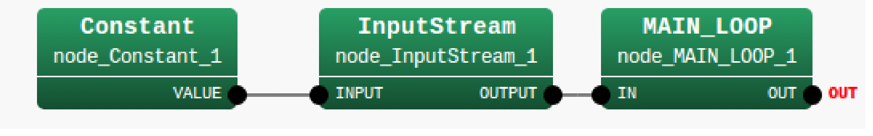


Figure 14.18: MAIN (subnet)

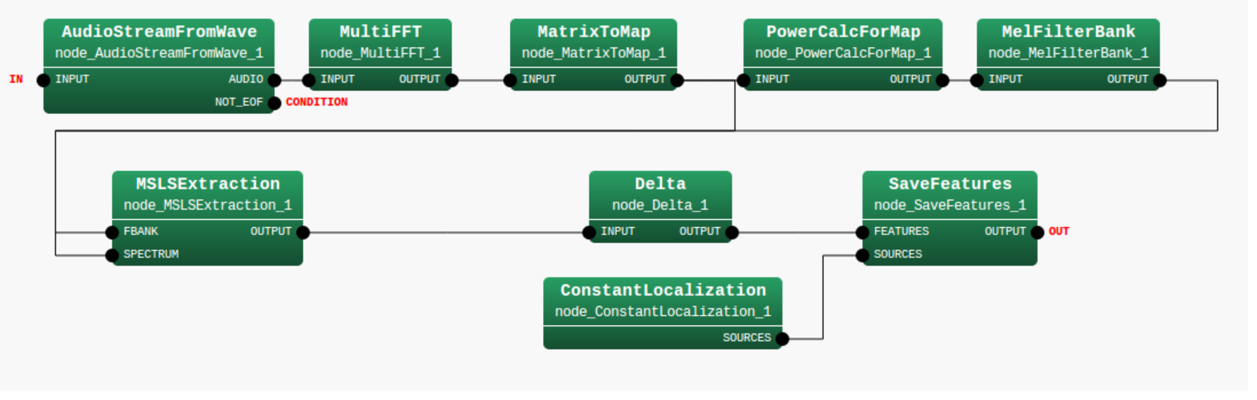


Figure 14.19: MAIN_LOOP (iterator)

Table 14.14 summarizes the main parameters

14.5.4 MSLS+Power

An execution example is shown in Figure 14.20. After the execution, a file named `MFBANK14_0.spec` is generated. This file stores little endian 14 dimensional vector sequence expressed in the 32 bit floating-

Table 14.14: Parameter list

Node name	Parameter name	Type	Value
MAIN_LOOP	LENGTH	subnet_param	int :ARG2
	ADVANCE	subnet_param	int :ARG3
	SAMPLING_RATE	subnet_param	int :ARG4
	FBANK_COUNT	subnet_param	int :ARG5
	DOWHILE	bool	(empty)
Delta	FBANK_COUNT	subnet_param	FBANK_COUNT

point number format. When separation cannot be performed well, check the following items, check if the f101b001.wav is in the ../data directory.

```
> ./demo.sh 3
 UINodeRepository::Scan()
 Scanning def /usr/lib/flowdesigner/toolbox
 done loading def files
 loading XML document from memory
 done!
 Building network : MAIN
```

Figure 14.20: Execution example

Twelve modules are included in this sample. There are three modules in MAIN_LOOP (iterator) and nine modules in MAIN (subnet). MAIN (subnet) and MAIN_LOOP (iterator) are shown in 14.21 and 14.22 As an outline of the processing, it is simple network configuration in which acoustic features are calculated in MSLSExtraction with the audio waveforms collected in the AudioStreamFromWave module and are written in SaveFeatures . Since MSLSExtraction requires the outputs of the mel-scale filter bank and power spectra for calculation of MSLS, the collected audio waveforms are analyzed by MultiFFT and their data type are converted by MatrixToMap and PowerCalcForMap , and then processing to obtain outputs of the mel-scale filter bank is performed by MelFilterBank . Here, the USE_POWER property of MSLSExtraction is set to `true`and to output the power term at the same time. MSLSExtraction reserves 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 USE_POWER 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 FBANK_COUNT property of MSLSExtraction +1 are output as a feature. Outputs other than the necessary MSLS coefficient and item power term are deleted in FeatureRemover . SaveFeatures saves the input FEATURE. The localization result from the front generated by ConstantLocalization is gave to SOURCES.

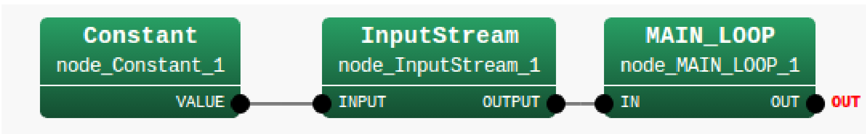


Figure 14.21: MAIN (subnet)

Table 14.15 summarizes the main parameters.

Table 14.15: Parameter list

Node name	Parameter name	Type	Value
MAIN_LOOP	LENGTH	subnet_param	int :ARG2
	ADVANCE	subnet_param	int :ARG3
	SAMPLING_RATE	subnet_param	int :ARG4
	FBANK_COUNT	subnet_param	int :ARG5
	DOWHILE	bool	(empty)
MSLSExtraction	FBANK_COUNT	subnet_param	FBANK_COUNT
	NORMALIZE_MODE	string	Cepstral
	USE_POWER	bool	true

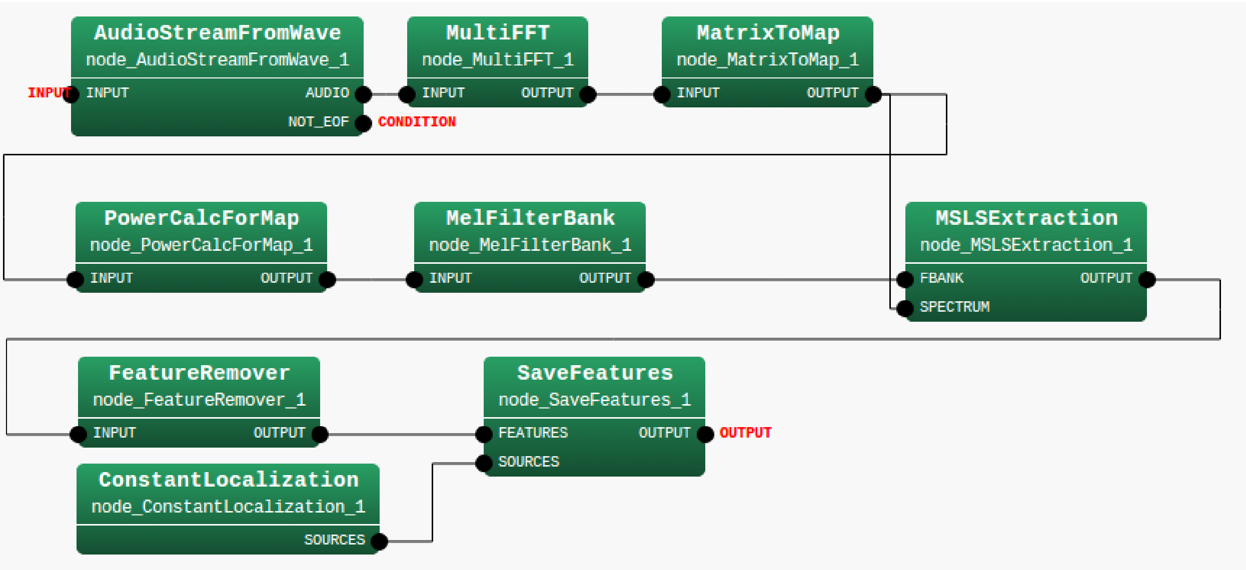


Figure 14.22: MAIN_LOOP (iterator)

14.5.5 MSLS+ΔMSLS+Power+ΔPower

Execute demo4.sh in the FeatureExtraction directory. An execution example is shown in Figure 14.23. After the execution, a file named `MFBANK28_0.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 if the f101b001.wav files are in the data directory.

```
> ./demo.sh 4
 UINodeRepository::Scan()
 Scanning def /usr/lib/flowdesigner/toolbox
 done loading def files
 loading XML document from memory
 done!
 Building network : MAIN
```

Figure 14.23: Execution example

Twelve modules are included in this sample is dozen. There are three modules in MAIN_LOOP (iterator) and nine modules in MAIN (subnet). MAIN (subnet) and MAIN_LOOP (iterator) are shown in 14.24 and 14.25. As an outline of the processing, it is simple network configuration in which acoustic features are calculated in MSLSExtraction with the audio waveforms collected in the AudioStreamFromWave module and are written in SaveFeatures . Since MSLSExtraction requires the outputs of the mel-scale filter bank and power spectra for calculation of MSLS, the collected audio waveforms are analyzed by MultiFFT and their data type are converted by MatrixToMap and PowerCalcForMap , and then processing to obtain outputs of the mel-scale filter bank is performed by MelFilterBank . Here, the USE_POWER property of MSLSExtraction is set to `true`and to output the power term at the same time. MSLSExtraction reserves 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 USE_POWER 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 FBANK_COUNT property of MSLSExtraction +1 are output as a feature. The δ MSLS coefficient and delta power term are calculated and stored with Delta . SaveFeatures saves the input FEATURE. The localization result from the front generated by ConstantLocalization is gave to SOURCES.
Table 14.16 summarizes the main parameters.

14.5.6 MSLS+ΔMSLS+ΔPower

An execution example is shown in Figure 14.26. After the execution, a file named `MFBANK27_0.spec` is generated. This file stores little endian 27 dimensional vector sequence expressed in the 32 bit floating-point

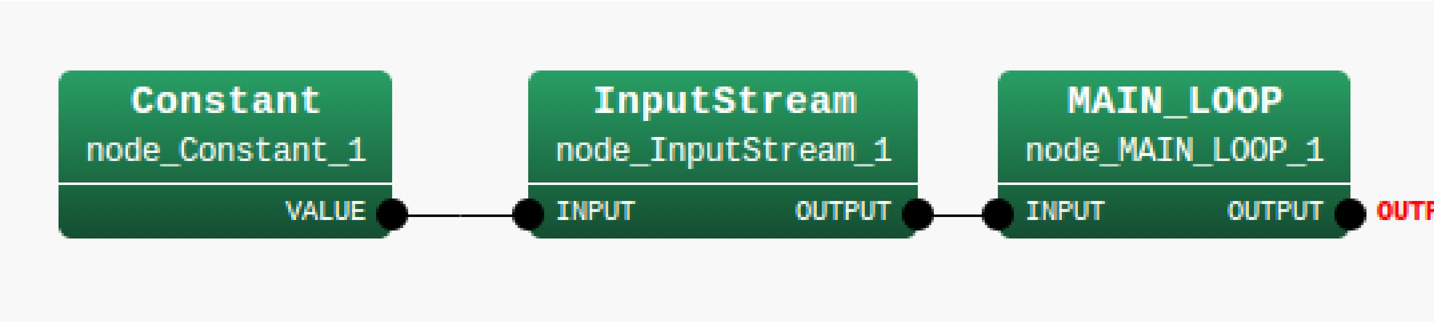


Figure 14.24: MAIN (subnet)

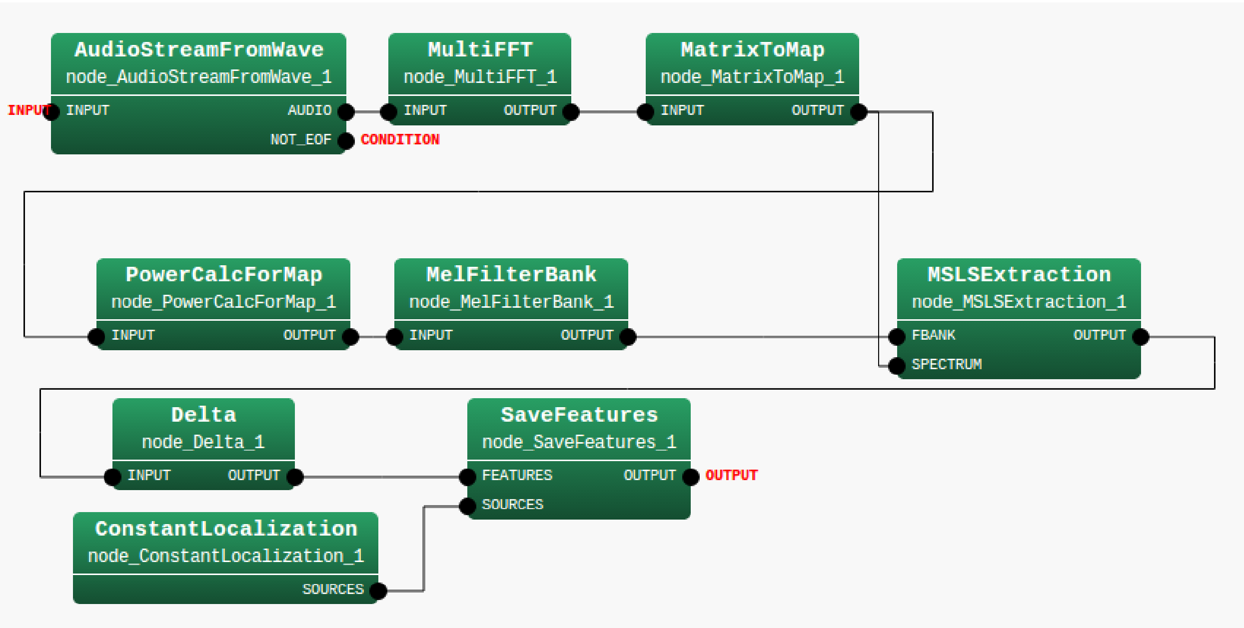


Figure 14.25: MAIN_LOOP (iterator)

Table 14.16: Parameter list

Node name	Parameter name	Type	Value
MAIN_LOOP	LENGTH	subnet_param	int :ARG2
	ADVANCE	subnet_param	int :ARG3
	SAMPLING_RATE	subnet_param	int :ARG4
	FBANK_COUNT	subnet_param	int :ARG5
	FBANK_COUNT1	subnet_param	int :ARG6
	DOWHILE	bool	(empty)
MSLSExtraction	FBANK_COUNT	subnet_param	FBANK_COUNT
	NORMALIZE_MODE	string	Cepstral
	USE_POWER	bool	true
Delta	FBANK_COUNT1	subnet_param	FBANK_COUNT1

number format. When feature extraction cannot be performed well, check if the f101b001.wav files are in the data directory.

Thirteen modules are included in this sample. There are three modules in MAIN_LOOP (iterator) and ten modules in MAIN (subnet). MAIN (subnet) and MAIN_LOOP (iterator) are shown in Figures 14.27 and 14.28 As an outline of the processing, it is simple network configuration in which acoustic features are calculated in MSLSExtraction with the audio waveforms collected in the AudioStreamFromWave module and are written in SaveFeatures . Since MSLSExtraction requires the outputs of the mel-scale filter bank and power spectra for calculation of MSLS, the collected audio waveforms are analyzed by MultiFFT and their

```
> ./demo.sh 5
 UINodeRepository::Scan()
 Scanning def /usr/lib/flowdesigner/toolbox
 done loading def files
 loading XML document from memory
 done!
 Building network :MAIN
```

Figure 14.26: Execution example

data type are converted by MatrixToMap and PowerCalcForMap , and then processing to obtain outputs of the mel-scale filter bank is performed by MelFilterBank . MSLSExtraction reserves 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 USE_POWER 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 FBANK_COUNT 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 FeatureRemover to delete them. SaveFeatures saves the input FEATURE. The localization result from the front generated by ConstantLocalization is gave to SOURCES.

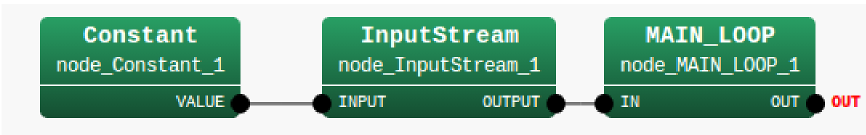


Figure 14.27: MAIN (subnet)

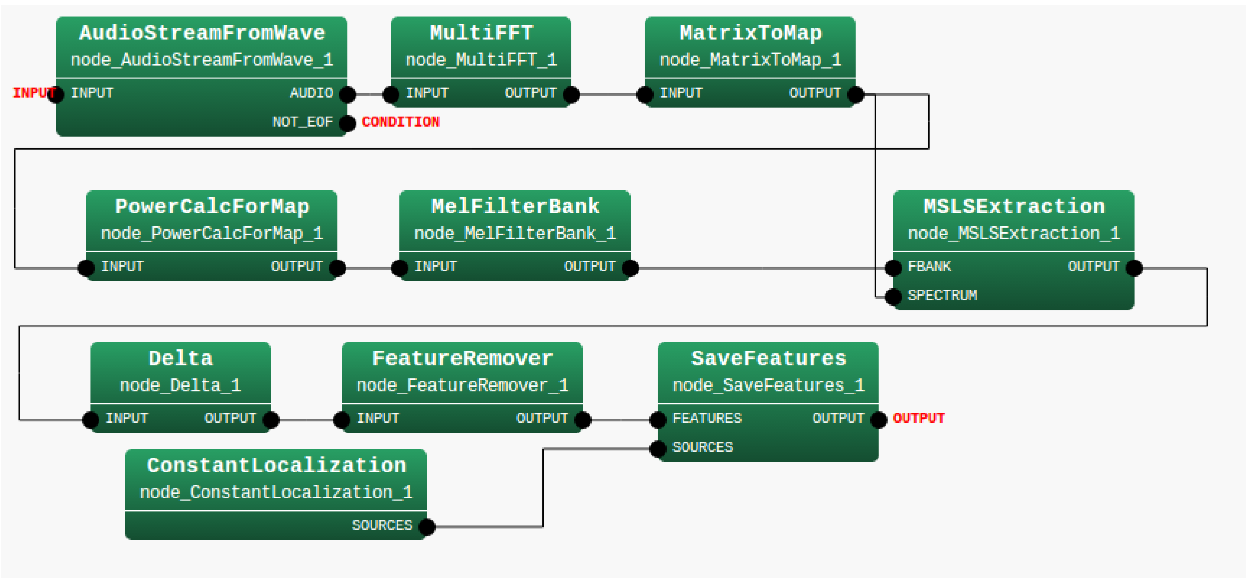


Figure 14.28: MAIN_LOOP (iterator)

Table 14.17 summarizes the main parameters.

14.5.7 MSLS+ΔMSLS+ΔPower+Preprocessing

An execution example is shown in Figure 14.29. After the execution, a file named `MFBANK27p_0.spec` is generated. This file stores little endian 27 dimensional vector sequence expressed in the 32 bit floating-point

Table 14.17: Parameter list

Node name	Parameter name	Type	Value
MAIN_LOOP	LENGTH	subnet_param	int :ARG2
	ADVANCE	subnet_param	int :ARG3
	SAMPLING_RATE	subnet_param	int :ARG4
	FBANK_COUNT	subnet_param	int :ARG5
	FBANK_COUNT1	subnet_param	int :ARG6
	DOWHILE	bool	(empty)
MSLSExtraction	FBANK_COUNT	subnet_param	FBANK_COUNT
	NORMALIZE_MODE	string	Cepstral
	USE_POWER	bool	true
Delta	FBANK_COUNT1	subnet_param	FBANK_COUNT1
FeatureRemover	SELECTOR	Object	<Vector<float> 13>

number format. When feature extraction cannot be performed well, check if the f101b001.wav is in the data directory.

```
> ./demo.sh 6
 UINodeRepository::Scan()
 Scanning def /usr/lib/flowdesigner/toolbox
 done loading def files
 loading XML document from memory
 done!
 Building network : MAIN
```

Figure 14.29: Execution example

Seventeen modules are included in this sample. There are three modules in MAIN_LOOP (iterator) and fourteen modules in MAIN (subnet). MAIN (subnet) MAIN (subnet) and MAIN_LOOP (iterator) are shown in Figures 14.30 and 14.31 As an outline of the processing, it is simple network configuration in which acoustic features are calculated in MSLSExtraction with the audio waveforms collected in the AudioStreamFromWave module 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 MatrixToMap and the signals are synthesized by Synthesize once. Pre-emphasis is performed for the synthesized waves with PreEmphasis , they are analyzed with MultiFFT once more, their type is converted with PowerCalcForMap and sent to MSLSExtraction . Since MSLSExtraction requires the outputs of the mel-scale filter bank and power spectra for calculation of MSLS, the collected audio waveforms are analyzed by MultiFFT and their data type are converted by MatrixToMap and PowerCalcForMap , and then processing to obtain outputs of the mel-scale filter bank is performed by MelFilterBank . MSLSExtraction reserves 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 USE_POWER 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 FBANK_COUNT 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 FeatureRemover to delete them. SaveFeatures saves the input FEATURE. The localization result from the front generated by ConstantLocalization is gave to SOURCES.

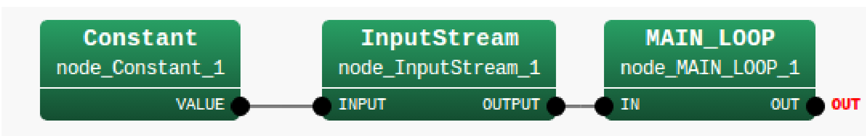


Figure 14.30: MAIN (subnet)

Table 14.18 summarizes the parameters of the network. Its main modules are PreEmphasis , MSLSExtraction , SpectralMeanNormalization , Delta , and FeatureRemover . see HARK document for details.

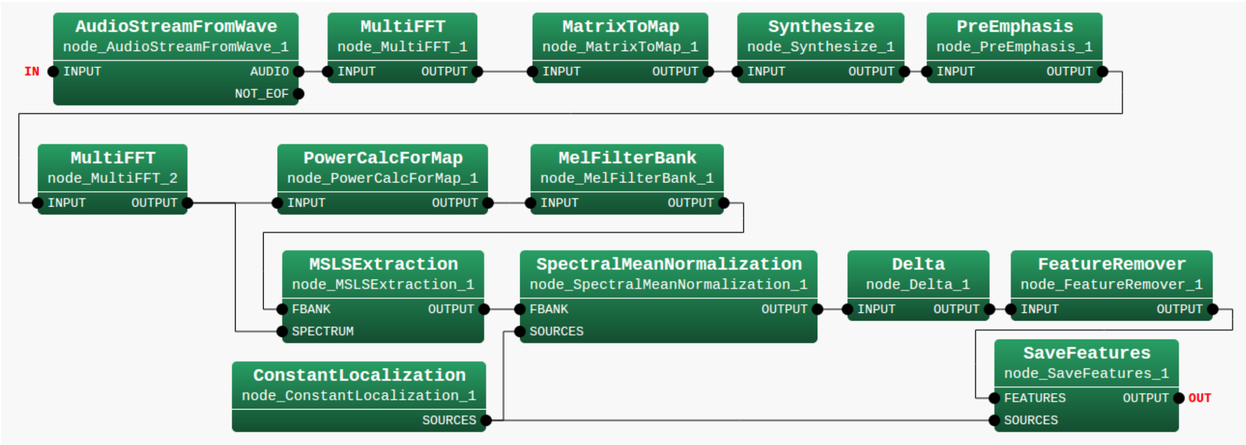


Figure 14.31: MAIN_LOOP (iterator)

Table 14.18: Parameter list

Node name	Parameter name	Type	Value
Constant	VALUE	subnet_param	int :ARG1
MAIN_LOOP	LENGTH	subnet_param	int :ARG2
	ADVANCE	subnet_param	int :ARG3
	SAMPLING_RATE	subnet_param	int :ARG4
	FBANK_COUNT	subnet_param	int :ARG5
	FBANK_COUNT1	subnet_param	int :ARG6
	DOWHILE	bool	(empty)
PreEmphasis	LENGTH	subnet_param	LENGTH
	SAMPLING_RATE	subnet_param	SAMPLING_RATE
	PREEMCOEFF	float	0.97
	INPUT_TYPE	string	WAV
MSLSExtraction	FBANK_COUNT	subnet_param	FBANK_COUNT
	NORMALIZE_MODE	string	Cepstral
	USE_POWER	bool	true
SpectralMeanNormalization	FBANK_COUNT1	subnet_param	FBANK_COUNT1
Delta	FBANK_COUNT1	subnet_param	FBANK_COUNT1
FeatureRemover	SELECTOR	Object	<Vector<float> 13>

14.6 Speech recognition network sample

This section introduces the sample including speech separation, recognition, and success rate evaluation. Although the samples are for off-line use, you can use it for online processing just replacing AudioStreamFromWave to AudioStreamFromMic . All sample files are in Recognition directory. See Table 14.6.1 for details. The rest of this section describes how to run the samples step-by-step.

Table 14.19: The list of files

Category	File name	Description
Data	../MultiSpeech_en.wav	Wave file used in this sample
JuliusMFT	julius.jconf	Configuration file of JuliusMFT
	AM/hmmdefs_en.bin	Acoustic model of English pronunciation
	AM/allTriphones_en	List of triphones in the acoustic model
	LM/order.*	Grammar-based language model
HARK	Recognition_en.n	HARK network file for localization, separation, and feature extraction
	Recognition.sh	Shell script to run the network file
	../config/microcone_tf.zip	Transfer function for localization and separation (for HARK 2.1.0 or later)
	sep_files	Directory for separated sounds
Evaluation	score.py	Evaluation script
	transcription_A.txt	Reference data of the utterances for each direction
	transcription_B.txt	Reference data of the utterances for each direction

14.6.1 Running the speech recognition

Prepare two terminals. Run speech recognition using JuliusMFT in one terminal, and run speech separation using HARK in another terminal. The commands and results are shown in Figs. 14.32 and 14.33, respectively. Note that you need run JuliusMFT first to make sure that the separated sound is sent after the initialization of JuliusMFT. After the separation, you will find wave files in sep_files/ directory, and a result.txt, a speech recognition log file.

```
> 1_Julius.sh
After you see the message "waiting client at 10500",
press enter again [Press Enter]

 <-- You press enter

STAT: include config: julius.jconf
STAT: loading plugins at "/usr/lib/julius_plugin":
STAT: file: calcmix_heu.jpi  #0 [Gaussian calculation plugin for Julius.
                                                        (ADD_MASK_TO_HEU)]
      ... skipped ...
///////////////////////////////
///  Module mode ready
///  waiting client at 10500
///////////////////////////////

 <-- You press enter again.
```

Figure 14.32: An execution example of JuliusMFT

The wave files are the separated sound to be recognized. Since these wave files are standard monaural audio files, you can listen to them by your audio player.

The text file result.txt is a raw speech separation log.

Trouble shooting

If no wave files are created, it means that HARK does not work correctly. Check if you have all files listed in Table , and sep_files/ directory is writable. Next, check if you successfully installed HARK (See the recipe 3.1Installation fails for this.)

If result.txt includes no recognition result, it means that JuliusMFT does not work. Check if JuliusMFT is installed. Run `julius_mft` in your terminal. If you see `command not found`, it is not installed yet. Next, check if all files listed in Table exist. Finally, check if you typed exactly the same command shown in Fig. 14.32.

For any case, the reason of the error will be written in log file or error messages. Read them carefully.

```
> 2_Recognition.sh
 UINodeRepository::Scan()
 Scanning def /usr/local/lib/flowdesigner/toolbox
 done loading def files
 loading XML document from memory
 done!
 Building network  :MAIN
 TF = 1,INITW = 0,FixedNoise = 0
 SSMethod = 2LC_CONST = 0LC_Method = 1
 reading A matrix
 72 directions, 8 microphones, 512 points
 done
 initialize
 Source 0 is created.
 Source 1 is created.
    ... skipped ...
```

Figure 14.33: An execution example of HARK

14.6.2 Evaluating the speech recognition

The next step is to evaluate the success rate of speech recognition using an evaluation script `score.py`.

```
> 3_Evaluation.sh
```

Each argument of the python script means that a speech recognition log, a reference data, a sound direction, and a tolerance.

After you run the script, you will see the result like Fig. 14.34. Starting from the left, each row means that the recognition is succeed or not, recognition result, and the reference. The last line means the overall success rate. In this case, 33 utterances out of 40 utterances are successfully recognized, consequently, the success rate us 85%.

ground truth	recognition result	status
"pork-cutlet-bowl"	""	Deletion
"curry-and-rice"	""	Deletion
"beef-bowl"	"beef-bowl"	Correct
"seafood-salad"	"seafood-salad"	Correct
"scrambled-eggs"	"scrambled-eggs"	Correct
	(skipped)	
"beef-bowl"	"beef-bowl"	Correct
33 / 40 (82.5 %)		

Figure 14.34: Recognition result.

For any directions, the success rates should be around 80%. If the rate is extremely low, check if you specified the correct pair of a direction and a reference data. If the rate is still low, the separation or recognition may fail. Listen to the files in sep_files/ to check if the separation is succeeded, or refer to the recipes in Chapter 3.