PyHARK: A HARK-based python package for robot audition and computational auditory scene analysis

Kazuhiro Nakadai[1,a)] Masayuki Takigahira[2] Katsutoshi Itoyama[1,2,b)]

Abstract: This paper describes PyHARK, a Python package for open source robot audition software HARK, which was newly introduced in HARK 3.4. There are two versions of PyHARK; online and offline. Both allow users to access HARK directly from Python. Online PyHARK maintains the small overhead of module integration, and offline PyHARK provides an easy interface to Python to make a script as easy as possible. Experimental results showed that online/offline PyHARK achieved real-time processing even under a strict condition where HARK does not work in real time.

Keywords: robot audition, HARK, Python, software package, real-time processing, online/offline processing

1. Introduction

HARK [1] has been being developped since 2008 as open source robot audition software [2], [3], [4], [5]. Generally, a GUI programming environment, HARKDesigner, is used to program on HARK. It is user-friendly for beginners, but is not efficient for expert programmers because a Web browser has to be run each time for HARKDesigner. To solve this problem, this paper proposes PyHARK, a Python package that allows HARK functions to be called directly from Python scripts. This means that common programming environments such as Jupiter Notebook [6] and Visual Studio Code [7] can be used. In addition, it supports both online and offline processing, so that it can be used in different situations.

2. The Architecture for PyHARK

Fig. 1a) shows the HARK 3.4 software stack, which consists of four layers: user program, HARK modules, harkmw, operating system. A HARK user program is called the "n" file and is an XML file that describes the connection relationships between HARK modules. At runtime, harkmw controls the flow of HARK modules written in C/C++ according to the description in the "n" file, as indicated by the arrow. Since harkmw integrates HARK modules only with C/C++ function calls, the integration overhead is maintained small. For PyHARK, two requirements were considered: One is to keep the overhead of HARK module integration small. This requires middleware such as harkmw. Otherwise, all data would have to be communicated between HARK modules via the user program (Python) layer, which requires time-consuming serialization and deserialization. Therefore, as shown in Fig. 1b), we constructed "online PyHARK," which inherits the advantages of HARK's online processing. This is achieved by wrapping the flow control function extracted from harkmw in Python, thus reducing the overhead of module integration. The other is the simplicity of programming. Since online PyHARK internally includes the flow control, the user needs to write the equivalent of an "n" file in Python. Because it is tedious to write it every time, we constructed "offline PyHARK" by removing harkmw, as shown in Fig. 1c). It allows users to call the HARK modules in an offline batch processing manner without the equivalent of an "n" file. Although it is not suitable for online processing due to serialization and deserialization, it is useful for prototyping because of simple programming.

[1] School of Engineering, Tokyo Institute of Technology.2-12-1-W8-30, Ookayama, Meguro, Tokyo, 152-8552, JAPAN
[2] Honda Research Institute Japan Co., Ltd., 8-1, Honcho, Wako, Saitama, 351-0188, JAPAN
[a)] nakadai@ra.sc.e.titech.ac.jp
[b)] itoyama@ra.sc.e.titech.ac.jp

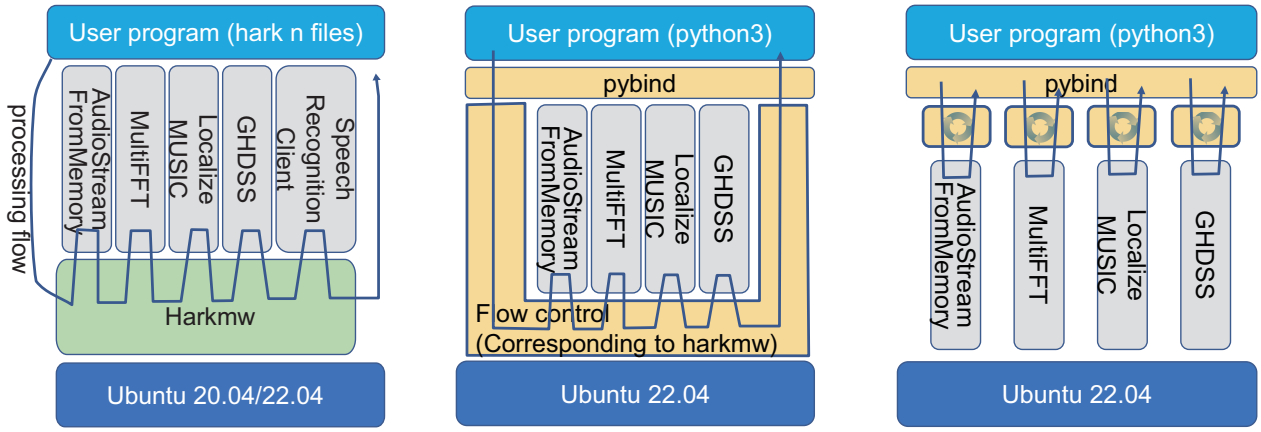
3. Implementation examples with PyHARK

Listing 1 shows a program for short time Fourier transform using a MultiFFT node of HARK for an 8 ch acoustic signal input by offline version of PyHARK.

Listing 1: pyhark-offline-sample.py

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import hark
import numpy

# number of microphones
nch   = 8

# generate input signal (8 ch, 32 bit float)
freqs = numpy.logspace(0, 3, endpoint=False, num=nch+1,
  base=2)[:nch] * 440
phase = numpy.arange(16000) * freqs[:, None] / 16000 * 2
   * numpy.pi
audio = numpy.sin(phase).astype(numpy.float32)

# framing (freme length 512, shift length 160)
input = numpy.lib.stride_tricks.sliding_window_view(
  audio, (nch, 512))[0, ::160]

multi_fft = hark.node.MultiFFT()
output = multi_fft(INPUT=input)

print(output.OUTPUT)
```

Listing 1 is free from the consideration of incremental process-



a) HARK 3.4 b) Online PyHARK c) Offline PyHARK

Fig. 1 Software Stacks in HARK and PyHARK (online & offline)

ing, and the program can be written simply. Lines 10-13 generate the input signal. Framing processing with the frame and shift length of 512 and 160 samples is performed for the generated signal in line 16. The framed data are then simply sent to the next node, that is, MultiFFT on lines 18 and 19. PyHARK also has function nodes for sound source localization and separation called LocalizeMUSIC and GHDSS[*1]. By feeding the output of MultiFFT to LocalizeMUSIC and GHDSS nodes as an input argument after their instanciation, sound source localization and separation can be achieved.

The online version of PyHARK requires more programing than the offline version to attain the same functionality. The online version provides incremental processing, and thus only the frame shift length data obtained from the microphone array are pushed to the publisher each time. However, it is not always possible to obtain exactly the frame length of data each time, and AudioStreamFromMemory having a buffer to absorb fluctuations in the amount of acquired data, is necessary for data management with buffer control. In addition, the online version requires implementation of classes corresponding to HARK networks called "n" file. The classes define the overall flow control and the processing flow of a single frame. Each class is constructed by placing functional nodes, setting the properties of each functional node, and connecting the nodes as in the HARK n-file.

PyHARK assumes that a protype program is first develped with the offline version to validate theoretical correctness, and the program is seamlessly migrated to the online version for an online demo or proof-of-concept. This reduces the load compared to writing in MATLAB or python and then re-implementing it in C/C++ or other online frameworks for demo purposes.

The embedded version, which will be presented as another paper, is designed to be highly compatible with the online version. This will be helpful in actual development for IoT and other embedded applications.

4. Evaluation

The processing time of online PyHARK, offline PyHARK, and HARK was evaluated through a task connecting three HARK modules, MultiFFT (short-time Fourier transform), Localize-

[*1] geometric high-order decorrelation-based source separation

Table 1 Processing Time

	HARK 3.4	Online PyHARK	Offline PyHARK
Mean	32.7	15.5	15.1
S.D	0.31	2.63	0.08

MUSIC (source localization with Multiple Signal Classification (MUSIC)), and SourceTracker (source tracking). An 8 ch wav file of 20 seconds duration that included two sound sources was used as input. The parameter of the frequency of Eigenvalue decomposition (EVD) for MUSIC was set to once per frame, instead of once every 50 frames which is normally used. Ubuntu 22.04 on VMWare Player 16 was used, with four Intel i7-12700K CPUs and 32 GB memory allocated. For each condition, the processing time was measured 10 times, and Tab. 1 shows the mean and standard deviation. Online and offline PyHARK had almost the same performance, and both maintained real-time processing as they were under 20 seconds. On the other hand, HARK 3.4 could not achieve real-time processing due to frequent EVD operations.

5. Conclusion

This paper introduced PyHARK which allows users to access HARK functionality both online and offline from Python. We showed that PyHARK achieved real-time processing under strict parameter settings where HARK 3.4 did not work in real-time. The future work includes the support of GPU, FPGA, and ARM processors.

Acknowledgments This work was supported by KAKENHI JP19KK0260, JP20H00475 and JP23K1116.

References

[1] "HARK," https://hark.jp/.
[2] K. Nakadai, T. Takahashi, H. G. Okuno, H. Nakajima, Y. Hasegawa, and H. Tsujino, "Design and implementation of robot audition system "HARK"," *Advanced Robotics*, vol. 24, pp. 739–761, 2010.
[3] K. Nakadai, H. G. Okuno, and T. Mizumoto, "Development, deployment and applications of robot audition open source software HARK," *Journal of Robotics and Mechatronics*, vol. 29, no. 1, pp. 16–25, 2017.
[4] K. Nakadai, H. G. Okuno, H. Nakajima, Y. Hasegawa, and H. Tsujino, "An open source software system for robot audition hark and its evaluation," in *2008 IEEE RAS International Conference on Humanoid Robots (Humanoids 2008)*, 2008, pp. 561–566.
[5] K. Nakadai, H. G. Okuno, T. Takahashi, K. Nakamura, T. Mizumoto, T. Yoshida, T. Otsuka, and G. Ince, "Introduction to open source robot audition software HARK," in *The 29th Annual Conference of the Robotics Society of Japan (RSJ2011)*, 2011.
[6] "Jupyter Notebook," https://jupyter.org/.
[7] "Visual Studio Code," https://azure.microsoft.com/ja-jp/products/visual-studio-code/.