NARA-WPE: A Python package for weighted prediction error dereverberation in Numpy and Tensorflow for online and offline processing

Lukas Drude, Jahn Heymann, Christoph Boeddeker, Reinhold Haeb-Umbach

Paderborn University, Department of Communications Engineering, Paderborn, Germany
Email: {drude, heymann, boeddeker, haeb}@nt.upb.de

Abstract

NARA-WPE is a Python software package providing implementations of the weighted prediction error (WPE) dereverberation algorithm. WPE has been shown to be a highly effective tool for speech dereverberation, thus improving the perceptual quality of the signal and improving the recognition performance of downstream automatic speech recognition (ASR). It is suitable both for single-channel and multi-channel applications. The package consist of (1) a Numpy implementation which can easily be integrated into a custom Python toolchain, and (2) a TensorFlow implementation which allows integration into larger computational graphs and enables backpropagation through WPE to train more advanced front-ends. This package comprises of an iterative offline (batch) version, a block-online version, and a frame-online version which can be used in moderately low latency applications, e.g. digital speech assistants.

1 Introduction

Background noise and signal reverberation due to reflections in an enclosure are the two main impairments in acoustic signal processing and far-field speech recognition. This work addresses signal dereverberation techniques based on WPE for speech recognition and other far-field applications.

WPE is a compelling algorithm to blindly dereverberate acoustic signals based on long-term linear prediction. Proposed as early as 2008 it gained constant attention in subsequent years with generalizations to multi-channel applications and more advanced speech source models [1–7]. High ASR performance gains were presented during the REVERB challenge [8–10] where WPE was able to improve results on top of an already competitive beamforming front-end [11]. Possibly most prominent is a recursive formulation used in the Google Home speech assistant hardware in online conditions [12, 13] although recursive formulations were already published much earlier [14, 15].

Traditionally, computationally expensive algorithms have been written in low-level languages like C or Fortran. With the rise of scripting languages which tend to be easier to learn and allow a more interactive user experience scientific computing has become more accessible. In particular, Python is widely accepted in the machine learning and signal processing community and offers a wide set of scientific computing libraries. Most notably, Numpy is a numerical Python library which efficiently manipulates large arrays and offers C and Fortran bindings to e.g. optimized linear algebra libraries [16, 17].

TensorFlow is a framework which allows to build up a computational graph which can then run on different nodes/ CPUs and GPUs [18]. It supports a large set of operations (ranging from elementary to more complicated linear algebra) to perform numerical computations. Gradients for a large subset of the operations are available which thus allows to perform algorithmic differentiation.

NARA-WPE now aims at extending the scientific library landscape by introducing an open source implementation of WPE dereverberation. To be precise, it currently consists of a highly optimized Numpy implementation of iterative WPE for offline and online processing and a TensorFlow implementation for offline, block-online and online processing as summarized in Tbl. 1. The complete code is publically available on Github[1] under the MIT license and can be installed with one of the following commands:

```
pip install nara_wpe
pip install git+https://github.com/fgnt/nara_wpe
```

Table 1: Content of the NARA-WPE package.

Framework	Offline	Block-Online	Online
Numpy	✓		✓
TensorFlow	✓	✓	✓

2 Background and motivation

When developing new algorithms, it is crucial to compare results to previously developed methods. Ideally, methods are accessible to reproduce results and modify code to build new tools and systems. To the best of our knowledge, no WPE implementation is publically available as of yet. However, an encrypted Matlab file `wpe.p`[2] can be used but does not encourage further research and code changes [2]. To facilitate and promote WPE usage a simple and easy to use Python software package is presented here.

3 Scenario and signal model

Using D microphones, we observe a signal which is represented as the D-dimensional vector $\mathbf{y}_{t,f}$ at time frame index t and frequency bin index f in the short time Fourier transformation (STFT) domain. In a far-field scenario, this signal is impaired by (convolutive) reverberation. We assume, that for perception and ASR the early part of the room impulse response (RIR) is beneficial whereas the reverberation tail deteriorates the recognition and should therefore be suppressed. Specifically, we consider the first 50 ms after the main peak of the RIR ($h^{(\text{early})}$) to contribute to the direct signal whereas the remaining part ($h^{(\text{tail})}$) is the cause of the distortions. In the STFT domain we denote this model as follows:

$$\mathbf{y}_{t,f} = \mathbf{x}_{t,f}^{(\text{early})} + \mathbf{x}_{t,f}^{(\text{tail})}, \tag{1}$$

where $\mathbf{x}_{t,f}^{(\text{early})}$ and $\mathbf{x}_{t,f}^{(\text{tail})}$ are the STFTs of the source signal which is convolved with the early part of the RIR and with the late reflections, respectively. To be precise, we explicitly allow RIRs longer than the length of a DFT window.

[1] https://github.com/fgnt/nara_wpe
[2] http://www.kecl.ntt.co.jp/icl/signal/wpe/

4 Weighted prediction error

The underlying idea of WPE is to estimate the reverberation tail of the signal and subtract it from the observation to obtain an optimal estimate of the "early" speech (which consists of the direct signal component and early reflections) in a maximum likelihood sense. This section provides an overview of two existing WPE variants.

Given filter weights $g_{\tau,f,d,d'}$, an estimate of the clean speech can be obtained:

$$\hat{x}^{(\text{early})}_{t,f,d} = y_{t,f,d} - \sum_{\tau=\Delta}^{\Delta+K-1} \sum_{d'} g^*_{\tau,f,d,d'} y_{t-\tau,f,d'}$$

$$\hat{\mathbf{x}}^{(\text{early})}_{t,f} = \mathbf{y}_{t,f} - \mathbf{G}_f^{\mathsf{H}} \tilde{\mathbf{y}}_{t-\Delta,f}. \quad (2)$$

The delay $\Delta > 0$ is introduced to avoid whitening of the speech source, K is the number of filter taps, d is the sensor index and $\mathbf{G}_f \in \mathbb{C}^{DK \times D}$ and $\tilde{\mathbf{y}}_{t-\Delta,f} \in \mathbb{C}^{DK \times 1}$ are stacked representations of the filter weights and the observations. Fig. 1 visualized which observations are used to predict the reverberation tail at frame index t. WPE maximizes the likelihood of the model under the assumption that the single channel direct signal is a realization of a zero-mean complex Gaussian with an unknown channel independent time-varying variance $\lambda_{t,f}$:

$$p(x^{(\text{early})}_{t,f,d}; \lambda_{t,f}) = \mathcal{CN}(x^{(\text{early})}_{t,f,d}; 0, \lambda_{t,f}). \quad (3)$$

The corresponding multichannel distribution is:

$$p(\mathbf{x}^{(\text{early})}_{t,f}; \lambda_{t,f}) = \mathcal{CN}(\mathbf{x}^{(\text{early})}_{t,f}; 0, \lambda_{t,f}\mathbf{I}). \quad (4)$$

where $\mathbf{I}$ is the identity matrix. It can be noticed that replacing identity matrix with an arbitrary time invariant covariance matrix does not change the solution of WPE.

There is no known close form solution to estimate the variance $\lambda_{t,f}$ and the filter coefficients $\mathbf{G}_f$ directly.

4.1 Iterative offline approach

The default way to solve this is to rely on an iterative procedure which alternates between estimating the time-varying variance λ_{tf} and the filter coefficients $\mathbf{G}_f$ [3, 19]:

$$\text{Step 1)} \quad \lambda_{t,f} = \frac{1}{(\delta+1+\delta)\,D} \sum_{\tau=t-\delta}^{t+\delta} \sum_{d} |\hat{x}^{(\text{early})}_{\tau,f,d}|^2, \quad (5)$$

$$\text{Step 2)} \quad \mathbf{R}_f = \sum_t \frac{\tilde{\mathbf{y}}_{t-\Delta,f} \tilde{\mathbf{y}}^{\mathsf{H}}_{t-\Delta,f}}{\lambda_{t,f}} \in \mathbb{C}^{DK \times DK}, \quad (6)$$

$$\mathbf{P}_f = \sum_t \frac{\tilde{\mathbf{y}}_{t-\Delta,f} \mathbf{y}^{\mathsf{H}}_{t,f}}{\lambda_{t,f}} \in \mathbb{C}^{DK \times D}, \quad (7)$$

$$\mathbf{G}_f = \mathbf{R}_f^{-1} \mathbf{P}_f \in \mathbb{C}^{DK \times D}. \quad (8)$$

The heuristic context of $(\delta+1+\delta)$ frames helps to improve the variance estimate in this iterative scheme [3]. It is common to use the observation $y_{\tau,f,d}$ as initialization for the estimated signal.

Since experiments have shown that the initialization is already fairly good, it is also possible to drop the iterations altogether and just rely on a single filter estimation step.

4.2 Non-iterative block-online approach

There are different options how a block-online approach can be defined. In this context, if the iterations are dropped, the block-online solution is straightforward and simply consists of applying Eqs. 5 – 8 and Eq. 2 to a signal block. To use past observations the correlation matrix and correlation vector can be updated recursively with a decaying window, where b is the block index and $\mathcal{T}_b$ are all time frame indices belonging to block b and α is a decay factor (e.g. 0.9999):

$$\mathbf{R}_{b,f} = \alpha \mathbf{R}_{b-1,f} + \sum_{t \in \mathcal{T}_b} \frac{\tilde{\mathbf{y}}_{t-\Delta,f} \tilde{\mathbf{y}}^{\mathsf{H}}_{t-\Delta,f}}{\lambda_{t,f}}, \quad (9)$$

$$\mathbf{P}_{b,f} = \alpha \mathbf{P}_{b-1,f} + \sum_{t \in \mathcal{T}_b} \frac{\tilde{\mathbf{y}}_{t-\Delta,f} \mathbf{y}^{\mathsf{H}}_{t-\Delta,f}}{\lambda_{t,f}}. \quad (10)$$

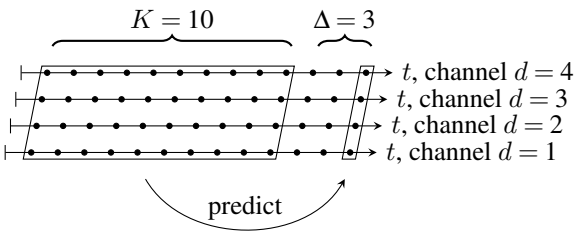

Figure 1: WPE estimates a filter to predict the current reverberation tail frame from K time frames which lie Δ time frames in the past. This frame (the reverberation tail) is then subtracted from the observed signal.

4.3 Non-iterative frame-online approach

In order to obtain an online variant with a minimum delay, the prediction is updated on the outdated filter coefficients $\mathbf{G}^{\mathsf{H}}_{t-1,f}$ first and then the correlation matrix and correlation vector are estimated in the same way as in the block-online case with a block size of 1:

$$\hat{\mathbf{x}}^{(\text{early})}_{t,f} = \mathbf{y}_{t,f} - \mathbf{G}^{\mathsf{H}}_{t-1,f} \tilde{\mathbf{y}}_{t-\Delta,f}, \quad (11)$$

$$\mathbf{R}_{t,f} = \alpha \mathbf{R}_{t-1,f} + \frac{\tilde{\mathbf{y}}_{t-\Delta,f} \tilde{\mathbf{y}}^{\mathsf{H}}_{t-\Delta,f}}{\lambda_{t,f}}, \quad (12)$$

$$\mathbf{P}_{t,f} = \alpha \mathbf{P}_{t-1,f} + \frac{\tilde{\mathbf{y}}_{t-\Delta,f} \mathbf{y}^{\mathsf{H}}_{t,f}}{\lambda_{t,f}}. \quad (13)$$

This leads to a recursive formulation with the following updates [13] where no matrix inversion and no calculation of $\mathbf{P}_{t,f}$ is necessary:

$$\mathbf{K}_{t,f} = \frac{\mathbf{R}^{-1}_{t-1,f} \tilde{\mathbf{y}}_{t-\Delta,f}}{\alpha \lambda_{t,f} + \tilde{\mathbf{y}}^{\mathsf{H}}_{t-\Delta,f} \mathbf{R}^{-1}_{t-1,f} \tilde{\mathbf{y}}_{t-\Delta,f}} \in \mathbb{C}^{DK \times 1}, \quad (14)$$

$$\mathbf{R}^{-1}_{t,f} = \frac{\mathbf{R}^{-1}_{t-1,f} - \mathbf{K}_{t,f} \tilde{\mathbf{y}}^{\mathsf{H}}_{t-\Delta,f} \mathbf{R}^{-1}_{t-1,f}}{\alpha} \in \mathbb{C}^{DK \times DK}, \quad (15)$$

$$\mathbf{G}_{t,f} = \mathbf{G}_{t-1,f} + \mathbf{K}_{t,f} \left(\hat{\mathbf{x}}^{(\text{early})}_{t,f}\right)^{\mathsf{H}} \in \mathbb{C}^{DK \times D}. \quad (16)$$

This is in essence a Recursive Least Squares (RLS) adaptive filter for the reverberation estimation. The authors [13] approximate Eq. 5 using only a left context δ_{L} with the observation directly:

$$\lambda_{t,f} = \frac{1}{(\delta_{\text{L}}+1)\,D} \sum_{\tau=t-\delta_{\text{L}}}^{t} \sum_{d} |y_{\tau,f,d}|^2. \quad (17)$$

5 Examples

This section provides a few code examples showcasing the high-level interface of NARA-WPE. The underlying code is modularized and allows to call the individual steps corresponding to the aforementioned equations independently.

5.1 Numpy iterative offline example

To dereverberate a multi-channel observation, you first need to load the data into a Numpy array such that the array has the shape `(channels, samples)`. You can then perform an STFT to obtain a Numpy array with the shape `(frequency bins, channels, time frames)`. Then, this is fed to the WPE function. An inverse STFT is then required to obtain a dereverberated result in time domain:

```
import numpy as np
from nara_wpe import np_wpe as wpe
from nara_wpe.utils import stft, istft

y = aquire_audio_data()
Y = stft(y).transpose(2, 0, 1)
Z = wpe(Y)
z = istft(Z.transpose(1, 2, 0))
```

At the time of writing, the interface supports the parameters listed in Tbl. 2. The `statistics_mode` governs how the beginning of an utterance is handled. If it is set to `'full'`, the signal is zero-padded such that an observation of T time frames yields T contributions to the sum in Eqs. 6 – 7. If it is set to `'valid'`, the vectors $\tilde{\mathbf{y}}_{t-\Delta}$ reaching out of the signal ($t - \Delta < 0$) will be ignored. For completeness reasons, Fig. 2 shows an example utterance before and after applying WPE.

5.2 TensorFlow iterative offline example

The iterative approach in TensorFlow uses the same API as the Numpy equivalent. It requires an observation with the shape `(frequency bins, channels, time frames)`. To keep the example concise we calculated the STFT in Numpy and just provided the result in the TensorFlow feed dictionary `{Y_tf: Y}`. An inverse STFT can then be applied to the resulting Numpy array `Z` to obtain a time domain signal.

```
import tensorflow as tf
from nara_wpe import tf_wpe as wpe
from nara_wpe.utils import stft, istft

y = aquire_audio_data()
Y = stft(y).transpose(2, 0, 1)
with tf.Session() as session:
    Y_tf = tf.placeholder(
        tf.complex128, shape=(None, None, None))
    Z_tf = wpe(Y_tf)
    Z = session.run(Z_tf, {Y_tf: Y})
z = istft(Z.transpose(1, 2, 0))
```

TensorFlow automatically distributes the processing steps on available CPUs and GPUs.

Table 2: Parameters of the Numpy and Tensorflow interface for the iterative offline implementation. See text for details.

Parameter	Symbol	Comment
`taps`	K	Number of filter taps in Eq. 2
`delay`	Δ	Delay in Eq. 2
`iterations`		Number of iterations
`psd_context`	δ	Smoothing context in Eq. 5
`statistics_mode`		Either `'full'` or `'valid'`.

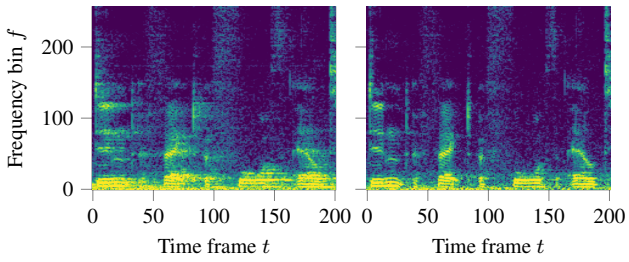

Figure 2: Power spectrum of an audio example before and after applying WPE (sampling rate 16 kHz, STFT size 512, STFT shift 128, color scaling from −60 dB to −20 dB).

5.3 TensorFlow frame-online example

A frame-online example requires, that certain state variables are kept from frame to frame. That is, the inverse correlation matrix $\mathbf{R}_{t,f}^{-1}$ is stored in `Q` and initialized with an identity matrix. The filter coefficients matrix is stored in `G` and initialized with zeros. As a consequence, the first frame prediction will just be the observation itself. Here, `aquire_framebuffer()` is a user provided function which yields the $K + \Delta$ most recent frames (compare Fig. 1) with a block shift of one frame. Then, the feed dictionary is prepared with the current framebuffer and the state variables.

```
import numpy as np
import tensorflow as tf
from nara_wpe.tf_wpe import online_wpe_step
from nara_wpe.utils import stft, istft

Q = np.identity(...)
G = np.zeros(...)
with tf.Session() as session:
    Y_tf = tf.placeholder(...)
    Q_tf = tf.placeholder(...)
    G_tf = tf.placeholder(...)
    results = online_wpe_step(Y_tf, Q_tf, G_tf)
    for Y in aquire_framebuffer():
        feed_dict = {Y_tf: Y, Q_tf: Q, G_tf: G}
        Z, Q, G = session.run(results, feed_dict)
```

Finally, `Z` holds the prediction for the current frame.

6 Back-propagation through WPE

TensorFlow provides an infrastructure to define a computational graph based on elementary operations. Besides the implementation of the operation itself (the forward step) many operations also include a backward step. This allows in particular, that gradients with respect to all inputs can be calculated for the provided TensorFlow implementation.

This enables a number of yet unexplored opportunities: A possible approach is to train a neural network to estimate the power spectrum similar to [20] with e.g. a signal reconstruction loss after the dereverberation step. Another possibility is to learn a smoothing matrix to generalize Eq. 5 or learn an optimal decay factor α.

However, it is worth noting that the gradient calculation is numerically problematic due to the matrix inversion (or equation solving) when estimating the filter matrix in Eq. 8. Experiments have shown that it is helpful to clip gradients if the magnitude exceeds a certain value or drop those gradients altogether. It is also helpful to train with a batch size larger than one such that the gradients are averaged over more observations. Finally, randomly sampling the WPE parameters (i.e. delay, number of filter taps and smoothing window) may help to avoid overfitting your model to a particular WPE parameter set.

7 Evaluation

To evaluate the implementation we first benchmark the frame-online implementation on our particular hardware. Then, we profile the Numpy implementation. Finally, we inspect the effect of the TensorFlow implementation on word error rate (WER) reduction in an ASR task.

7.1 Real-time factor of online implementation

Fig. 3 shows the real time factor of the frame-online WPE implementation in TensorFlow on a Raspberry Pi Model 3b+ device [21], which hosts a recent Raspbian Stretch installation including a Kernel version 4.14. The Pi is equipped with 1 GB LPDDR2 RAM memory and an ARM A53 quad-core processor running at 1.4 GHz. The experiment generates random frames and feeds these into the update algorithm such that the time to acquire a new frame is not taken into account. The real time factor exhibits some kind of quadratic dependency of filter taps K and at least up to $K = 10$ filter taps online processing is possible on this particular hardware for $D \leq 4$ channels.

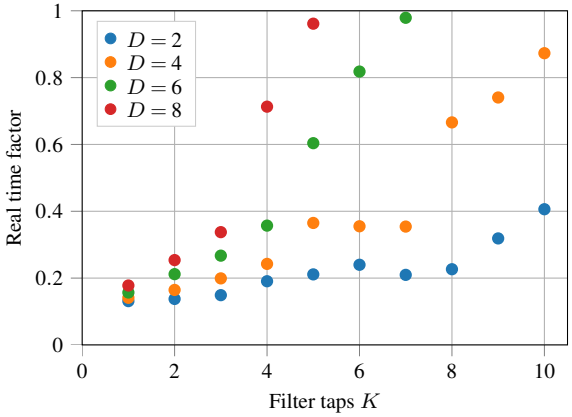

Figure 3: Real time factor for different parameter settings when running the frame-online WPE implementation on a Raspberry Pi Model 3b+ (1 GB LPDDR2 RAM, 1.4 GHz). Each dot represents an average of at least 100 runs.

7.2 Profiling of the Numpy implementation

The Numpy implementation of the multi-channel WPE algorithms is heavily optimized with respect to speed. In the current implementation, most processing time is spend on the estimation of the covariance matrix ($\approx 40\,\%$) and covariance vector ($\approx 10\,\%$). The second biggest fraction of time ($\approx 20\,\%$) is spent on the solver, which avoids a matrix inverse to obtain the filter matrix in the sense of Eq. 8. The main speed-up (in the region of a factor of 6) is achieved by using a so-called view[3] on the memory instead of creating copies of the input data. It is also worth noting that (at least in the current implementation) a loop over the independent axis (frequency axis) in Python is faster than operating on larger matrices in Numpy itself (at least a factor of 3). This is possibly caused by caching effects/ memory locality. Overall, the current implementation is at least a factor of 100 faster than a primitive Python implementation of the very generalized formulas presented in [3]. In contrast to [3] we use a scaled identity covariance matrix for the anechoic speech model in Eq. 4.

[3] https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.lib.stride_tricks.as_strided.html

7.3 Dereverberation effectiveness

To evaluate the effectiveness of the dereverberation implementation, we demonstrate ASR results on the REVERB challenge data [8]. It contains simulated training utterances and simulated and real test utterances. In particular, we compare an offline, block-online and online system in terms of WERs when using an offline wide residual network [22] as an acoustic model on the real recordings for testing. All model parameters were adapted from [22] and the acoustic model (AM) was trained on the multicondition training data. All systems are tested with a DFT window size of 512 (32 ms), shift of 128 (8 ms) and a Hann window. For all variants the WPE parameters are optimized on the development set (i.e. $\Delta \in \{1, \ldots, 4\}$, $K \in \{5, 10\}$) and averaged WERs for the test set are presented in Tbl. 3 taken from [23].

It can be observed that the offline iterative implementation improves the WER in both the 2 and 8 channel setup. This is particularly noteworthy and shows that even in times of very strong AMs a solid front-end is still beneficial. Simple smoothing of the power spectral density of the observed signal as a power estimate $\lambda_{t,f}$ and avoiding iterations altogether is much less computational expensive but still shows consistent gains in particular for 8 channels over the unprocessed baseline.

Table 3: WERs /% evaluated on the REVERB challenge real evaluation dataset averaged over near and far results. "Iteration" is the vanilla WPE according to Eqs. 5 – 8 whereas "Smoothing" uses Eq. 17 for PSD estimation and avoids iterations altogether.

	Offline		Block-Online		Online	
	2 ch	8 ch	2 ch	8 ch	2 ch	8 ch
Unprocessed	17.6	17.6	17.6	17.6	17.6	17.6
Iteration	**14.4**	**10.9**				
Smoothing	16.1	13.0	15.7	14.0	17.4	16.2

8 Conclusions

This document presents the NARA-WPE Python package with the aim to facilitate research in the field of multi-channel signal dereverberation and enable others to compare their results and extend or modify our code freely. We decided to publish both a Numpy and a TensorFlow implementation for offline and online applications to cover a wide range of possible future research directions. The source code is available on Github and an updated documentation is shipped with the current release code.

9 Acknowledgements

This work was in part supported by Deutsche Forschungsgemeinschaft (DFG) under contract no. Ha3455/11-1 and a Google Faculty Research Award. We thank Dr. Kinoshita, Dr. Nakatani and his colleagues for valuable discussions. Computational resources were provided by the Paderborn Center for Parallel Computing.

References

[1] T. Nakatani, T. Yoshioka, K. Kinoshita, M. Miyoshi, and B.-H. Juang, "Blind speech dereverberation with multi-channel linear prediction based on short time fourier transform representation," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2008.

[2] T. Nakatani, T. Yoshioka, K. Kinoshita, M. Miyoshi, and B.-H. Juang, "Speech dereverberation based on variance-normalized delayed linear prediction," *Transactions on Audio, Speech, and Language Processing*, 2010.

[3] T. Yoshioka and T. Nakatani, "Generalization of multi-channel linear prediction methods for blind MIMO impulse response shortening," *Transactions on Audio, Speech, and Language Processing*, 2012.

[4] A. Jukic and S. Doclo, "Speech dereverberation using weighted prediction error with Laplacian model of the desired signal," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2014.

[5] A. Cohen, G. Stemmer, S. Ingalsuo, and S. Markovich-Golan, "Combined weighted prediction error and minimum variance distortionless response for dereverberation," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 446–450, IEEE, 2017.

[6] N. Ito, S. Araki, and T. Nakatani, "Probabilistic integration of diffuse noise suppression and dereverberation," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2014.

[7] A. Jukic, T. van Waterschoot, T. Gerkmann, and S. Doclo, "Speech dereverberation with multi-channel linear prediction and sparse priors for the desired signal," in *Hands-free Speech Communication and Microphone Arrays (HSCMA), 2014 4th Joint Workshop on*, pp. 23–26, IEEE, 2014.

[8] K. Kinoshita, M. Delcroix, T. Yoshioka, T. Nakatani, A. Sehr, W. Kellermann, and R. Maas, "The REVERB challenge: A common evaluation framework for dereverberation and recognition of reverberant speech," in *Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, IEEE, 2013.

[9] M. Delcroix, T. Yoshioka, A. Ogawa, Y. Kubo, M. Fujimoto, N. Ito, K. Kinoshita, M. Espi, S. Araki, T. Hori, *et al.*, "Strategies for distant speech recognitionin reverberant environments," *EURASIP Journal on Advances in Signal Processing*, 2015.

[10] M. Delcroix, T. Yoshioka, A. Ogawa, Y. Kubo, M. Fujimoto, N. Ito, K. Kinoshita, M. Espi, S. Araki, T. Hori, *et al.*, "Defeating reverberation: Advanced dereverberation and recognition techniques for hands-free speech recognition," in *Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 522–526, IEEE, 2014.

[11] M. Delcroix, T. Yoshioka, A. Ogawa, Y. Kubo, M. Fujimoto, N. Ito, K. Kinoshita, M. Espi, T. Hori, T. Nakatani, and A. Nakamura, "Linear prediction-based dereverberation with advanced speech enhancement and recognition technologies for the REVERB challenge," in *Proc. REVERB Challenge Workshop*, vol. 1, pp. 1–8, 01 2014.

[12] B. Li, T. Sainath, A. Narayanan, J. Caroselli, M. Bacchiani, A. Misra, I. Shafran, H. Sak, G. Pundak, K. Chin, *et al.*, "Acoustic modeling for Google Home," *Interspeech*, 2017.

[13] J. Caroselli, I. Shafran, A. Narayanan, and R. Rose, "Adaptive multichannel dereverberation for automatic speech recognition," in *Interspeech*, 2017.

[14] T. Yoshioka, H. Tachibana, T. Nakatani, and M. Miyoshi, "Adaptive dereverberation of speech signals with speaker-position change detection," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3733–3736, IEEE, 2009.

[15] T. Yoshioka and T. Nakatani, "Dereverberation for reverberation-robust microphone arrays," in *European Signal Processing Conference (EUSIPCO)*, IEEE, 2013.

[16] P. F. Dubois, K. Hinsen, and J. Hugunin, "Numerical Python," *Computers in Physics*, vol. 10, May/June 1996.

[17] P. F. Dubois, "Extending Python with Fortran," *Computing Science and Engineering*, vol. 1, pp. 66–73, Sep/Oct 1999.

[18] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning.," in *OSDI*, vol. 16, pp. 265–283, 2016.

[19] C. Boeddeker, "Optimization of multi-channel dereverberation techniques for noisy reverberant speech recognition," Master's thesis, Paderborn University, 2018.

[20] K. Kinoshita, M. Delcroix, H. Kwon, T. Mori, and T. Nakatani, "Neural network-based spectrum estimation for online WPE dereverberation," *Interspeech*, 2017.

[21] Raspberry Pi, "https://www.raspberrypi.org/," ARM documentation, 2018.

[22] J. Heymann, L. Drude, and R. Haeb-Umbach, "Wide residual BLSTM network with discriminative speaker adaptation for robust speech recognition," in *CHiME-4 workshop*, 2016.

[23] J. Heymann, L. Drude, R. Haeb-Umbach, K. Kinoshita, and T. Nakatani, "Frame-online DNN-WPE dereverberation," in *Under review for International Workshop on Acoustic Signal Enhancement (IWAENC)*, 2018.