Hardware-Oriented Memory-Limited Online Artifact Subspace Reconstruction (HMO-ASR) Algorithm

Lan-Da Van, *Senior Member, IEEE*, You-Cheng Tu, Chi-Yuan Chang, Hsiang-Jen Wang, and Tzyy-Ping Jung, *Fellow, IEEE*

***Abstract*—Artifact Subspace Reconstruction (ASR) is a machine learning technique widely used to remove non-brain signals (referred to as "artifacts") from electroencephalograms (EEGs). The ASR algorithm can, however, be constrained by the limited memory available on portable devices. To address this challenge, we propose a Hardware-Oriented Memory-Limited Online ASR (HMO-ASR) algorithm. The proposed HMO-ASR algorithm consists of (1) two-level window-based preprocessing including PCA-based and z-score-based preprocessing to clean the data in each window, (2) iterative mean, standard deviation, and covariance update using a parallel algorithm to achieve window-based processing, and (3) early eigenvector matrix determination to save the computation. With the three schemes, the HMO-ASR method can be implemented on mobile devices, application-specific integrated circuits (ASICs) or field programmable gate arrays (FPGAs) with limited memory. The study results showed that the proposed HMO-ASR algorithm can achieve comparable performance to those obtained by the offline ASR algorithm with a 98.64% reduction in memory size. An FPGA implementation is used for silicon proof of the proposed HMO-ASR algorithm.**

***Index Terms*—Artifact subspace reconstruction, hardware-oriented, limited memory, and EEG.**

I. Introduction

ELECTROENCEPHALOGRAM (EEG) plays an important role in neuroscience research, clinical diagnosis [1], and mental state assessment [2]. Nevertheless, EEG recordings are vulnerable to pervasive contamination from non-brain signals, i.e., eye movements, muscle activities, etc., known as "artifacts." Hence, artifact removal is an essential step during EEG analyses. Artifact subspace reconstruction (ASR) [3]–[5] is one of the popular and effective artifact-removal methods. ASR can be used to enhance the signal quality before independent component analysis (ICA) [6]–[9]. However, the ASR algorithm requires considerable memory size, making it inadequate for performing online artifact removal on portable devices, application-specific integrated circuits (ASICs) or field programmable gate arrays (FPGAs). As the demands of daily health monitoring grow, the disadvantage of the ASR algorithm becomes more problematic. To match the performance of the offline ASR algorithm, this study developed a hardware-oriented and memory-constrained online ASR (HMO-ASR) algorithm. The rest of the paper is organized as follows. Section II discusses the proposed HMO-ASR algorithm. Section III discusses the results of the simulations and implementation. Finally, a remark concludes this work.

Manuscript received September 23, 2021; accepted October 17, 2021. Date of publication November 1, 2021; date of current version November 24, 2021. This work was supported in part by the Ministry of Science and Technology under Grant MOST 109-2221-E-009-084-MY3 and Grant MOST PAIR LABs; and in part by the National Science Foundation under Grant NCS-1734883, Grant CBET-1935860, Grant IP-1719130, and Grant SMA-1540943. This brief was recommended by Associate Editor Y. Ha and E. Bonizzoni. *(Corresponding authors: Chi-Yuan Chang; Tzyy-Ping Jung; Lan-Da Van.)*

This work involved human subjects or animals in its research. Approval of all ethical and experimental procedures and protocols was granted by the Institutional Review Board of the University of California, San Diego (190706).

Lan-Da Van, You-Cheng Tu, and Hsiang-Jen Wang are with the Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu 300, Taiwan (e-mail: ldvan@cs.nctu.edu.tw).

Chi-Yuan Chang and Tzyy-Ping Jung are with the Department Bioengineering and Swartz Center for Computational Neuroscience, University of California at San Diego, San Diego, CA 92093 USA (e-mail: chc418@ucsd.edu; tpjung@ucsd.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSII.2021.3124253.

Digital Object Identifier 10.1109/TCSII.2021.3124253

II. The MHO-ASR Algorithm

The HMO-ASR algorithm is an online artifact-removal algorithm that aims to reduce memory requirement without sacrificing effectiveness. It contains three main parts: (1) the two-level window-based preprocessing, (2) the rejection threshold calibration processing, and (3) reconstruction module. Fig. 1 shows the flow chart of the HMO-ASR algorithm. The two-level window-based preprocessing (in the upper dash-line block) features a PCA-based and a moving z-score based preprocessing modules to generate and select a cleaner window, respectively. The rejection threshold calibration processing (in the lower dash-line block) adopts an iterative updating scheme and an early eigenvector matrix determination module to update the corresponding rejection thresholds. The reconstruction module removes the principal components (PC) with values greater than the rejection threshold and reconstructs the data using the remaining PCs. Finally, the non-overlapped reconstructed samples are outputted and then the next new input window can be processed.

A. Two-Level Window-Based Preprocessing

Offline ASR requires a set of clean sections, where the data contain no artifacts, to determine the artifact removal threshold. ASR first performs PCA on the whole recording. Next, ASR calculates channel-wise root-mean-square (RMS) values with a non-overlapping sliding window and transforms the values into z-score. Finally, ASR selects the windows with z-score in range $-3 < z < 5$ [5] and defines them as clean

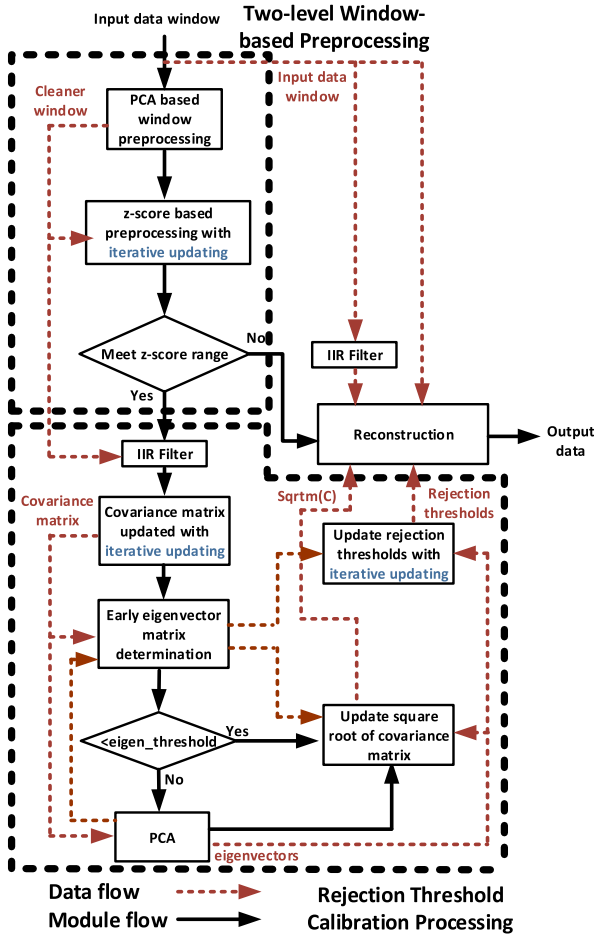


Fig. 1. Main flow chart of the proposed HMO-ASR algorithm.

sections. To utilize the information in the data and maximize the advantage of online algorithm, we propose to make ASR artifact rejection threshold adaptive. Since a contaminated data will significantly slow down the convergence speed of ASR artifact rejection threshold, the incoming data should be selected before updating the threshold. In our proposed two-level window-based preprocessing, we first perform PCA on the incoming window. In the PCA-based window processing module, we transform the eigenvalues into the z-score. If the eigenvalue based z-score is above 1.5, the corresponding eigenvector will be removed. The eigenvalue based z-score is defined as

$$z_\lambda = \frac{\lambda_k - m_\lambda}{\sigma_\lambda} \tag{1}$$

where λ_k is the k^{th} sorted eigenvalue, and m_λ and σ_λ are the mean and standard deviation of the eigenvalues, respectively. The preserved eigenvectors are then projected back to the channel domain. Next, in the z-score based preprocessing with iterative updating module, if the incoming window has a z value between -3 and 5, it will be sent to the rejection threshold calibration processing to update the artifact removal threshold. Otherwise, it will be marked as contaminated and sent to the reconstruction module.

B. Iterative Mean, Standard Deviation, Covariance Matrix Update

To save memory, we use a window-based approach to update the mean, standard derivation, covariance matrix, z-score based preprocessing, and rejection threshold iteratively. Assume an incoming window $\mathbf{x}_* = \{x_{*,1}, \ldots, x_{*,n_*}\}$ at iteration i has n_* samples, four statistical variables including the total number of samples n_{i-1}, the overall mean m_{i-1}, the sample standard deviation s_{i-1}, and the sample covariance matrix $\mathbf{C}_{i-1}$ are needed to be updated while reaching the i^{th} iteration. According to the parallel algorithm [10]-[12], we can derive m_*, s_*, and c_* respectively, for each channel across samples of $\mathbf{x}_*$ in an iterative updating way at the i^{th} iteration as follows:

$$n_i = n_{i-1} + n_* \tag{2}$$

$$\delta = m_* - m_{i-1} \tag{3}$$

$$m_i = m_{i-1} + \delta\frac{n_*}{n_i} = m_* - \delta\frac{n_{i-1}}{n_i} \tag{4}$$

$$\begin{aligned} s_i &= \sqrt{\frac{\sum_{p=1}^{n_{i-1}}\left(x_{i-1,p} - m_i\right)^2 + \sum_{q=1}^{n_*}\left(x_{*,q} - m_i\right)^2}{n_{i-1}}} \\ &= \sqrt{\frac{(n_{i-1}-1)s_{i-1}^2 + (n_*-1)s_*^2 + \frac{\delta^2 n_* n_{i-1}}{n_i}}{n_{i-1}}} \end{aligned} \tag{5}$$

Considering two arbitrary channels $\mathbf{a}, \mathbf{b}$ that have samples $\{a_1, \ldots, a_{n_{i-1}}\}$ and $\{b_1, \ldots, b_{n_{i-1}}\}$, respectively, with new samples $\{a_{n_{i-1}+1}, \ldots, a_{n_{i-1}+n_*}\}$ and $\{b_{n_{i-1}+1}, \ldots, b_{n_{i-1}+n_*}\}$ for the two channels, the covariance matrix in an iterative updating way is derived as follows:

$$\begin{aligned} c_i &= \frac{\sum_{p=1}^{n_{i-1}+n_*}(a_p - m_{a,i})(b_p - m_{b,i})}{n_{i-1}+n_*-1} \\ &= \frac{1}{n_i - 1}[(n_{i-1}-1)c_{i-1} + (n_*-1)c_* \\ &\quad + \frac{n_{i-1}n_*(m_{a,i-1} - m_{a,*})(m_{b,i-1} - m_{b,*})}{n_i}] \end{aligned} \tag{6}$$

According to Eqs. (4)-(6), it only requires parameters from the $(i-1)^{th}$ iteration to update parameters at the i^{th} iteration. Therefore, we can achieve window-based processing.

C. Early Eigenvector Matrix Determination

Calculating the square root of the covariance matrix ($sqrtm(\mathbf{C_i})$) at each iteration is computationally intensive. Moreover, the benefit of it is little if the eigenvectors of the incoming data and the previous data are similar. Hence, we propose to define a threshold to evaluate whether an eigenvalue decomposition is required for updating the $sqrtm(\mathbf{C_i})$.

Let $\mathbf{C}_i$ and $\mathbf{E}_{i-1}$ denote the updated covariance matrix and the previous eigenvector matrix, respectively. Since $\mathbf{C}$ is real-symmetric and diagonalizable, we can derive:

$$\mathbf{E}_{i-1}^T\mathbf{C}_i\mathbf{E}_{i-1} = \mathbf{E}_{i-1}^T\mathbf{E}_i\mathbf{D}_i\mathbf{E}_i^T\mathbf{E}_{i-1} \tag{7}$$

where the off-diagonal elements will be close to zero if $\mathbf{E}_i$ is close to $\mathbf{E}_{i-1}$ since $\mathbf{E}_i^T\mathbf{E}_{i-1} \approx \mathbf{I}$. Based on this characteristic,

we define the threshold as

$$\frac{\sum off\text{-}diag(\mathbf{E}_{i-1}^{T}\mathbf{C}_{i}\mathbf{E}_{i-1})}{\sum diag(\mathbf{E}_{i-1}^{T}\mathbf{C}_{i}\mathbf{E}_{i-1})} < eigen\text{-}threshold \tag{8}$$

where the eigen-threshold is empirically determined. We use 0.002 for the eigen-threshold for the following analyses. If the incoming data satisfies Eq. (8), HMO-ASR retains $\mathbf{E}_{i-1}$ to approximate the eigenvector matrix $\mathbf{E}_i$ of $sqrtm(\mathbf{C}_i)$. Otherwise, eigenvalue decomposition is required. The eigenvector matrix $\mathbf{E}_i$ of $sqrtm(\mathbf{C}_i)$ determines the artifact removal threshold and the reconstruction process.

D. Threshold Update and Reconstruction

The procedures of determining the artifact-removal threshold in offline ASR are as follows: ASR first filters the clean sections of the data with an IIR-filter to get ($\mathbf{X}_c$) and calculates the eigenvector matrix $\mathbf{E}_c$ of the square root of the covariance matrix of $\widetilde{\mathbf{X}}_c$. The IIR filter is used to emphasize the frequencies where artifacts commonly appear. Next, $\widetilde{\mathbf{X}}_c$ are projected onto the principal component space $\widetilde{\mathbf{Y}}_c = \mathbf{E}_c^T \cdot \widetilde{\mathbf{X}}_c$. Then, $\widetilde{\mathbf{Y}}_c$ are splited into several windows and the RMS values for principal components are computed within the window. Finally, the mean μ_c and standard deviation σ_c of the RMS values across all the windows are computed and a fixed rejection threshold Γ_c is defined as $\Gamma_c = \mu_c + f \cdot \sigma_c$, where f is the adjustable cutoff parameter. In the HMO-ASR algorithm, once the incoming data $\widetilde{\mathbf{X}}_*$ satisfy the z-score range, the HMO-ASR updates the rejection thresholds. Firstly, HMO-ASR follows the procedures above (with eigenvector matrix described in Section II.C) and calculates μ_* and σ_*. Next, the overall μ_i and σ_i are updated by the iterative updating scheme. Since $\mathbf{E}_i$ and $\mathbf{E}_{i-1}$ may be different, we project μ_{i-1} and σ_{i-1} onto $\mathbf{E}_i$ before performing the update. Finally, an adaptive threshold is defined as

$$\Gamma_i = \mu_i + f \cdot \sigma_i \tag{9}$$

After updating the rejection threshold, the contaminated data $\mathbf{X}_j$ will enter the reconstruction module. First, $\mathbf{X}_j$ is filtered by an IIR filter and is divided into multiple sub-windows $\widetilde{\mathbf{X}}_k$. Second, for each $\widetilde{\mathbf{X}}_k$, PCA is applied to obtain $\mathbf{C}_k = \mathbf{E}_k\mathbf{D}_k\mathbf{E}_k^T$. Third, compare $\lambda_{k,l}$ and the rejection thresholds Γ_i projected from $\mathbf{E}_i$ onto $\mathbf{E}_k$, the inequality equation can be repeated as:

$$\lambda_{k,l} > \sum_{p} \left(\Gamma_p (\mathbf{e}_i)_p^T (\mathbf{e}_k)_l\right)^2 \tag{10}$$

where $\lambda_{k,l}$ is the l^{th} element in $diag(\mathbf{D}_k)$. If the inequality holds, the corresponding principal components $(\mathbf{e}_k)_l$ are replaced with zero vectors, and we obtain a projection matrix $\left(\mathbf{E}_k^T sqrtm(\mathbf{C}_i)\right)_{trunc}$ and reconstruct $\mathbf{X}_j$ by:

$$(\mathbf{X}_k)_{clean} = sqrtm(\mathbf{C}_i)\left(\mathbf{E}_k^T sqrtm(\mathbf{C}_i)\right)_{trunc}^{\dagger}\mathbf{E}_k^T\mathbf{X}_k \tag{11}$$

where † denotes a pseudo inverse operator. Once all sub-windows are proceeded, HMO-ASR can free the memory used by $\mathbf{X}_j$.

III. Results and Discussion

This section presents study results and hardware verification.

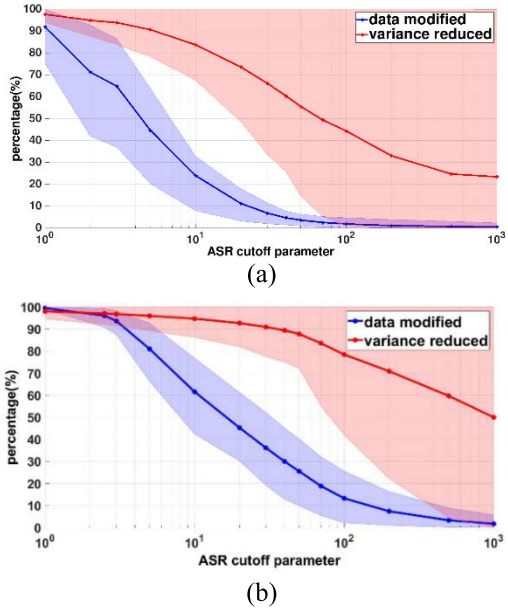


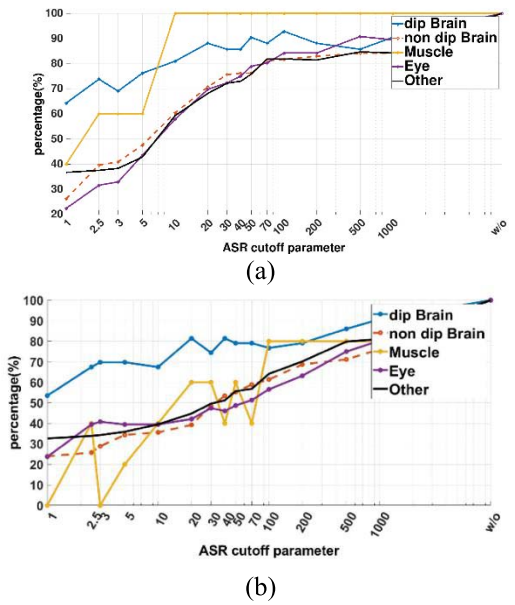
Fig. 2. Data modified and variance reduction. (a) HMO-ASR and (b) offline ASR.

A. Experiment and Data Acquisition

To evaluate the proposed algorithm, we used EEG recordings in a cued-artifact experiment. Ten healthy, college-aged subjects participated in the cued-artifact experiments in a purely voluntary manner, after providing informed written consents, under experimental protocols approved by the Institutional Review Board of the University of California, San Diego (#190706). For each subject, 30-channel EEG data were recorded using Cognionics Quick-30 dry EEG system at 500Hz sampling rate. There are two sessions in the experiment, in which we cue subjects to perform specific movements. In the eye-related cued-artifact session, subjects performed blinks and saccades. We asked subjects to reduce head/body movements throughout the session. In the motion-related cued-artifact session, subjects performed head-turning, nodding, and walking in the same place. The recorded data are down-sampled to 250Hz and bad channels are removed for each dataset for the simulation in Section III.B and implementation in Section III.D.

B. Performance Comparison

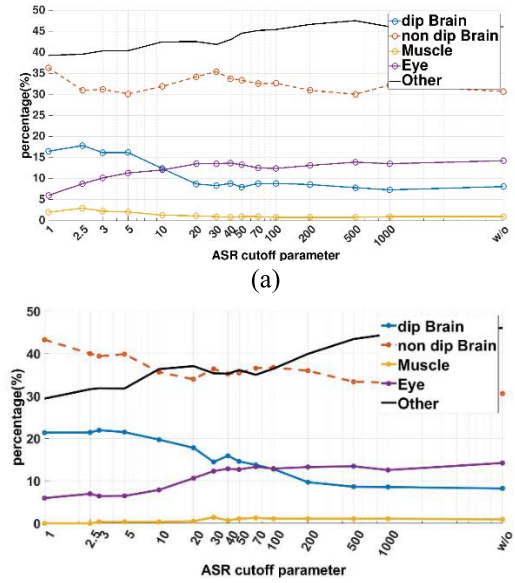
We follow the methods used in [5] to compare the performance of the proposed HMO-ASR and the offline ASR. We first characterize how the algorithm modifies data. Next, we investigate whether the algorithm preserves brain signals and removes artifacts. To separate brain signals from noise, we use an IC classifier called *ICLabel*, [13], a plug-in of EEBLAB [14]. Finally, we evaluate how the algorithm improves the quality of ICA decompositions. The correlation coefficient we used to determine whether an IC is preserved is 0.8 and the dipolarity we used to define a dipolar IC is 95. Both classification and dipolarity calculations were processed using EEGLAB plug-in functions. Fig. 2 shows that the HMO-ASR with cutoff parameter $f = 10$ modifies 25% of data and reduces 85% of the variance, which approached the performance of the offline ASR with $f = 50$.



(a)

(b)

Fig. 3. The percentage of preserved ICs within each class after ASR. (a) HMO-ASR and (b) offline ASR.



(a)

(b)

Fig. 4. The percentage of ICs in each class after ASR. (a) HMO-ASR and (b) offline ASR.

Next, we compare the percentage of preserved ICs within each class after the HMO-ASR and offline ASR. Fig. 3(a) shows that the HMO-ASR preserves 80% of dipolar brain ICs and removes 40% of IC from other classes when using $f = 10$. This result is comparable to the performance of offline ASR with $f = 50$ as shown in Fig. 3(b). Finally, we compare the percentage of ICs in each class after the HMO-ASR and offline ASR. Fig. 4(a) shows that the percentage of dipolar brain ICs increases from 8% to 16% and the percentage of eye ICs drops from 15% to 11% after HMO-ASR cleaning with $f = 5$. On the other hand, Fig. 4(b) shows that offline ASR has comparable performance when using $f = 20$. In summary, the empirical results show that the HMO-ASR with f between 5 and 10 can achieve comparable performance to the offline ASR with f between 20 to 50, which covers the suggesting range for the offline ASR [5]. These results suggested that the performance of our HMO-ASR can approach the optimal performance of offline ASR with limited memory.

TABLE I
FPGA RESOURCE UTILIZATION

Resource	Utilization	Available	Utilization (%)
LUT	100946	230400	43.81
LUT RAM	4291	101760	4.22
FF	115904	460800	25.15
BRAM	197.5	312	63.3
URAM	35	96	36.46
DSP	866	1728	50.12

C. Memory Size Comparison

Since the offline ASR needs to store the overall raw data in the memory, according to our experimental setting in Section III-A, the approximate memory size as an example of 30 channels is:

$$\begin{aligned} A_{M,Offline\text{-}ASR} &= 30(channels) \\ &\quad \times 1397.848(second) \\ &\quad \times 250(sample\ rate) \times 32bits \\ &= 41935.44\ KByte \end{aligned} \tag{12}$$

where 32 is the bits of floating points. Considering the proposed HMO-ASR algorithm (c.f. Fig. 1), according to our experimental setting in Section III-A, the approximate memory size is profiled and calculated as follows: (1) The memory size of two-level window-based preprocessing is estimated as 60.24 KByte. (2) The memory size of rejection threshold calibration processing is estimated as 51.72 Kbyte. (3) The memory size of the reconstruction module is estimated as 99.32 KByte. (4) The global temporary memory for the above-mentioned processes (1), (2) and (3) is estimated as 360 KByte. Hence, the total approximate memory size of the proposed HMO-ASR is

$$\begin{aligned} A_{M,HMO\text{-}ASR} &= 60.24 + 51.72 + 99.32 + 360 \\ &= 571.28\ KByte \end{aligned} \tag{13}$$

Comparing the memory sizes of Eqs. (12) and (13), the proposed HMO-ASR algorithm significantly reduces the memory size by 98.64% while obtaining satisfactory quality as shown in Figs. 2–4, allowing us to deploy the proposed algorithm to portable or edge devices for healthy monitoring in near real-time is possible.

D. FPGA Implementation

Hardware implementation uses Xilinx Zynq UltraScale + MPSoC ZCU104 for the silicon proof of HMO-ASR. We use high-level synthesis (HLS) to synthesize the C++ code and deploy it to the FPGA board. We adopt IEEE 754 floating-point single precision for the computation with a clock rate of 150 MHz. Table I shows the utilization of FPGA resources. The used memory, including both BRAM and URAM, is about

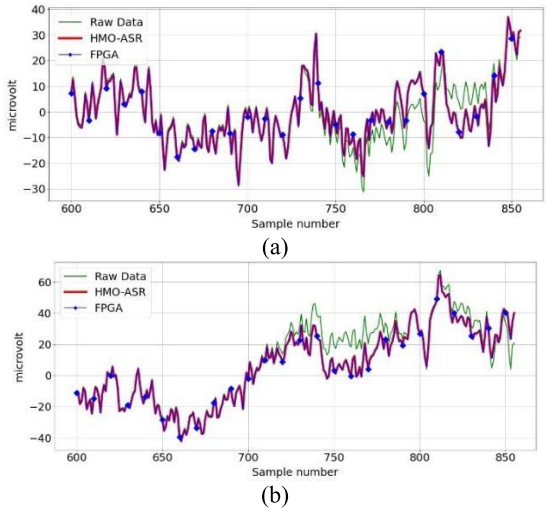


Fig. 5. Verification results (a) s07 eye channel 13 (b) s07 eye channel 6.

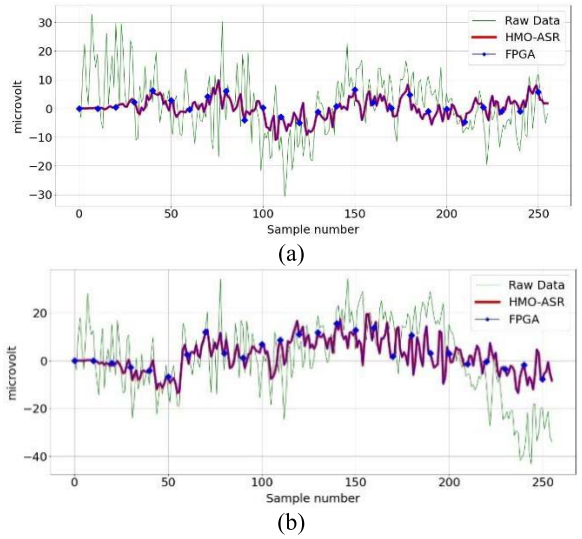


Fig. 6. Verification results (a) s03 motion channel 4 (b) s03 motion channel 6.

2.0975 MB. As compared to Eq. (13), the reason for the increased memory size is the use of more temporary registers and memory for computation. Note that this number is still much less than that in Eq. (12). We collected the first 12 seconds of recording for each dataset to test the FPGA implementation. Test signals were EEG signals contaminated by eye and motion artifacts from 10 subjects. Figs. 5 and 6 present the raw EEG data, along with the results of the HMO-ASR algorithm and the FPGA implementation of this algorithm. To demonstrate the performance of the proposed algorithm on a variety of datasets, Fig. 7 shows the root mean square error (RMSE) distributions based on the HMO-ASR algorithm and FPGA implementation. Because of the precision, a small portion of the FPGA output signals does not match the HMO-ASR algorithm output for datasets such as s01, s05, and s09 motion.

IV. Conclusion

This work proposed an online ASR solution (HMO-ASR) for hardware-oriented memory-limited devices (such as

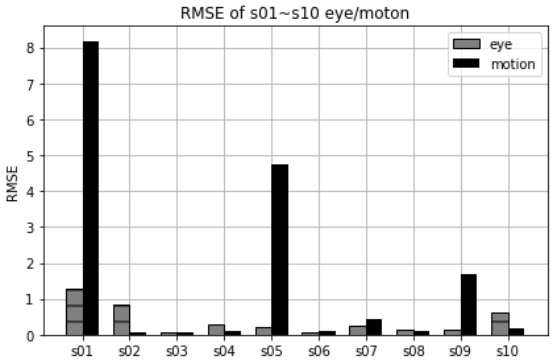


Fig. 7. RMSE performance across 20 EEG datasets.

edge devices) and ICs/FPGAs. The solution integrates two-level window-based preprocessing, mean/standard deviation/covariance iterative updating, and early eigenvector matrix determination schemes. Experimental results demonstrate the proposed HMO-ASR algorithm with cutoff parameter 5 to 10 exhibits comparable performance with the original offline ASR algorithm with cutoff parameter 20 to 50. The HMO-ASR algorithm can be efficiently implemented with FPGA hardware.

References

[1] E. D. Übeyli, "Statistics over features: EEG signals analysis," *Comput. Biol. Med.*, vol. 39, no. 8, pp. 733–741, Aug. 2009.

[2] A. L. Tandle, M. S. Joshi, A. S. Dharmadhikari, and S. V. Jaiswal, "Mental state and emotion detection from musically stimulated EEG," *Brain Informat.*, vol. 5, no. 2, p. 14, 2018.

[3] T. R. Mullen *et al.*, "Real-time neuroimaging and cognitive monitoring using wearable dry EEG," *IEEE Trans. Biomed. Eng.*, vol. 62, no. 11, pp. 2553–2567, Nov. 2015.

[4] C. A. E. Kothe and T. P. Jung, "Artifact removal techniques with signal reconstruction," U.S. Patent U.S. 20 160 113 587 A1, Apr. 2016.

[5] C.-Y. Chang, S.-H. Hsu, L. Pion-Tonachini, and T.-P. Jung, "Evaluation of artifact subspace reconstruction for automatic artifact components removal in multi-channel EEG recordings," *IEEE Trans. Biomed. Eng.*, vol. 67, no. 4, pp. 1114–1121, Apr. 2019.

[6] T.-W. Lee and T. J. Sejnowski, "Independent component analysis for mixed sub-Gaussian and super-Gaussian sources," in *Proc. 4th Joint Symp. Neural Comput.*, 1997.

[7] L.-D. Van, D.-Y. Wu, and C.-S. Chen, "Energy-efficient FastICA implementation for biomedical signal separation," *IEEE Trans. Neural Netw.*, vol. 22, no. 11, pp. 1809–1822, Nov. 2011.

[8] L.-D. Van, P.-Y. Huang, and T.-C. Lu, "Cost-effective and variable-channel FastICA hardware architecture and implementation for EEG signal processing," *J. Signal Process. Syst.*, vol. 82, no. 1, pp. 91–113, 2016.

[9] L.-D. Van, T.-C. Lu, T.-P. Jung, and J.-F. Wang, "Hardware-oriented memory-limited online FastICA algorithm and hardware architecture for signal separation," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, Brighton, U.K., 2019, pp. 1438–1442.

[10] D. H. D. West, "Updating mean and variance estimates: An improved method," *Commun. ACM*, vol. 22, no. 9, pp. 532–535, Sep. 1979.

[11] T. F. Chan, G. H. Golub, and R. J. LeVeque, "Updating formulae and a pairwise algorithm for computing sample variances," in *Proc. COMPSTAT 5th Symp. Held Toulouse*, 1982, pp. 30–41.

[12] "Algorithms for Calculating Variance." 2021. [Online]. Available: https://en.m.wikipedia.org/wiki/Algorithms_for_calculating_variance

[13] L. Pion-Tonachini, K. Kreutz-Delgado, and S. Makeig, "ICLabel: An automated electroencephalographic independent component classifier, dataset, and website," *NeuroImage*, vol. 198, pp. 181–197, Sep. 2019.

[14] A. Delorme and S. Makeig, "EEGLAB: An open source toolbox for analysis of single-trial EEG dynamics including independent component analysis," *J. Neurosci. Methods*, vol. 134, no. 1, pp. 9–21, 2004.