Embedded Streaming Principal Components Analysis for Network Load Reduction in Structural Health Monitoring

Alessio Burrello, *Student Member, IEEE*, Alex Marchioni, *Student Member, IEEE*, Davide Brunelli, *Senior Member, IEEE*, Simone Benatti, Mauro Mangia, *Member, IEEE*, and Luca Benini, *Fellow, IEEE*

***Abstract*—Principal Component Analysis (PCA) is a well-established approach commonly used for dimensionality reduction. However, its computational cost and memory requirements hamper the adoption of PCA in heavily resource-constrained embedded platforms. Streaming approaches have been proposed that may enable embedded implementations of the PCA. Among them, the History PCA (HPCA) algorithm stands out for its robustness to the variability in parameters and accuracy. This paper presents a parallel and memory-efficient implementation of HPCA in a structural health monitoring (SHM) application based on a heterogeneous network with sensor nodes measuring three-axial accelerations and gateways collecting measurements from several nodes and sending them to the cloud storage and analytic facility. In the targeted application, standard PCA reaches 15$\times$ compression factor with an average reconstruction signal to noise ratio of 16 dB and a negligible impact on the accuracy in the tracking of structural modal frequencies. By embedding HPCA on our SHM network gateways, we achieve the same compression factor as standard PCA, with more than 1000$\times$ reduction in data memory footprint for running the algorithm. Furthermore, we parallelize HPCA on the gateway, and we achieve a speedup of 7.1$\times$ (on 8 cores). Finally, we explore a fixed-point HPCA implementation on sensors (network end-nodes), that maximally distributes compression workload, minimizes required communication bandwidth, and maintains the same quality of reconstruction as HPCA in floating-point, with a compression factor of 10$\times$.**

***Index Terms*—Embedded platforms, Streaming PCA, Structural Health Monitoring, Edge computing, IoT**

I. Introduction

THE Internet of Things (IoT) envisions billions of devices that can sense, compute, and potentially communicate with users or among them [1]. Therefore, it poses new challenges in finding innovative and scalable approaches for collecting and processing the potentially massive amount of data. Even though a cloud platform [2] often plays the role of the central unit in the network that interconnects the different devices, the distribution of the processing to the edge of the cloud has demonstrated several advantages such as improved security, reduced latency and lower costs [3].

A. Burrello, A. Marchioni, D. Brunelli, M.Mangia, S. Benatti and L. Benini are with the Department of Electrical, Electronic and Information Engineering, University of Bologna, 40136 Bologna, Italy.
E-mail: alessio.burrello@unibo.it, alex.marchioni@unibo.it, simone.benatti@unibo.it, mauro.mangia2@unibo.it
D. Brunelli is also with the Department of Industrial engineering, University of Trento, 38123 Trento, Italy.
E-mail: davide.brunelli@unitn.it
L. Benini is also with the Department of Information Technology and Electrical Engineering at the ETH Zurich, 8092 Zurich, Switzerland.
E-mail: lbenini@iis.ee.ethz.ch
doi: 10.1109/JIOT.2020.3027102

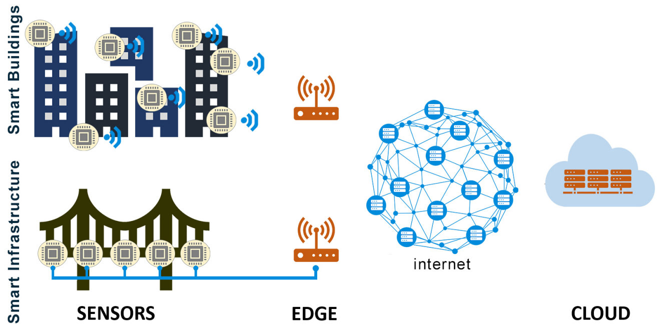

Fig. 1. Block scheme of IoT systems monitoring civil infrastructures and buildings where links between sensor nodes and edge device could be both wired o wireless.

Many attempts have been made to distribute workloads among network nodes using solely software [4], [5] and hardware-software codesign [6], or to address the trade-off between decentralization and single central computing unit [7]. Other solutions focus on handling the data traffic generated in these networks [8] to increase the number of connected devices, by optimizing the available bandwidth [9], [10], by increasing the performance of the network transfer infrastructure [9] or by compressing the data transmitted [11].

A leading IoT application is Structural Health Monitoring (SHM) [12], [13] whose aim is to provide a continuous flow of information about civil structure's condition. Here, continuous and long-term monitoring is gaining traction [14] for damage detection, predictive maintenance or even road traffic identification [15]. Figure 1 reports the conceptual block diagram of a SHM application. Despite the availability of low-cost sensor nodes (e.g. MEMS accelerometers), the sensor networks installed to acquire the data needed for big structure diagnosis are often forced to be sparse to limit data traffic [16]. However, spatially sparse monitoring has a limited precision in complex structures such as masonry buildings [17], or bridges [18], which demand extremely accurate damage detection and localization.

Therefore, researchers turn to compression algorithms to reduce the amount of data sent by the network nodes and to allow the deployment of more dense networks, enabling the fine-grain mapping of most critical structures. Compression algorithms can be categorized as lossless or lossy. Lossless approaches, such as [19], ensure complete information recovery, with a low compression rate. At the same time, lossy algorithms allow much higher compression rates, but they imply an application-dependent trade-off between compression and quality of the signal recovery. In a SHM scenario, the adoption of lossy compression algorithms is preferred, since the sensor readings are often aggregated in high-level features [20] whose accuracy can be even preserved in case of high compression rate [21], [22]. Noteworthy, the choice of an appropriate compression algorithm must also cope with possible limitations imposed by the devices involved in data compression (e.g. low onboard memory and compute capabilities).

The PCA-based compression algorithms are promising in the SHM field since i) they are not computationally intensive (the compression involves linear projections on a set of predefined sequences), and ii) they reach high compression ratio while preserving most important input signal features [22]. A critical shortcoming of these solutions is represented by the amount of data required to execute the PCA algorithm, which could impair their embedding on end-devices such as smart sensors and gateways. PCA streaming approaches [23] help to reduce the required memory footprint, since they calculate the principal components on smaller data chunks.

In this paper, we consider the History PCA (HPCA) algorithm, a streaming PCA approach characterized by higher robustness to variability in parameters compared to other streaming methods [24], [25]. We try to solve the high load network problem of SHM sensor-networks, combining this compression method with an efficient real-time implementation (edge-node computing). The proposed approach allows us to maintain a high-level accuracy in the SHM data-analysis (refer to Sec. VI-A), while it minimizes the network traffic. In addition, it could also be used as a standalone anomaly detection algorithm, as shown in [26], [27]. In particular, we used HPCA for compression in an SHM system composed of many sensors measuring three-axial acceleration and a few gateways (one gateway usually manages 40 to 50 sensors) employed to collect the sensor readings and send them to the cloud platform used for storage and processing. We embedded the algorithm on the gateway as well as on the sensors. The compression reduces the bandwidth of the connection to the cloud and the cloud storage space on gateways. The advantages are even higher on the sensor-nodes, where the reduced network traffic allows a higher node count per gateway. To profile our solutions and compare the deployments of HPCA algorithms, we use energy consumption and delay as figures of merit. We select the best solutions based on the minimization of energy. Indeed, minimizing the energy budget on a sensor network or, in general, on an edge computing platform, entails several advantages such as deploying a higher number of sensors on the same power bus, without increasing the maximum output current. On the other hand, the delay due to the processing is only related to the near real-time constraint since we have to process a batch of data before a new one is stored (see Sec. VI-C). The main contributions of the work are as follows:

1) We present an embedded sensor network deployed for SHM, based on HPCA data compression. To the best of our knowledge, this is the first attempt to embed a PCA based algorithm on a large-scale sensor network on low cost embedded platforms.
2) We show an extensive comparison with state-of-the-art compression methods, analyzing benefits and flaws of streaming PCA approaches and, in detail, of HPCA.
3) We validate our approach by showing how we are capable, on top of a tendon strand break, to detect a shift in the bridge resonant frequency using HPCA compressed signals, with a negligible accuracy loss compared to the detection obtained from the analysis of non-compressed data.
4) We explore the performance of HPCA both at gateway and sensor node level, by using energy and execution time to select the best implementations. At the gateway level, we consider two different platforms: ARTIK 710 Module [28] and Raspberry Pi 3 [29] and two different parallelization schemes of HPCA. On the sensor node, we propose a fixed-point solution that allows performing PCA directly on the tightly memory-constrained MCU (microcontroller unit – an STM32F405RG) as well as a floating-point single-core implementation.
5) We further investigate how different configurations of the HPCA algorithm could satisfy different memory constraints and the trade-off in network load reduction of moving PCA from gateways to sensors. We show that we can save up to $1221\times$ memory compared to the standard PCA on the gateway, and $1551\times$ on the sensor-nodes, reducing by $15\times$ and $10\times$ the traffic on networks links, respectively.

The rest of the article is organized as follows: Sec. II introduces the related works on compression algorithms. Sec. III presents the QR-decomposition (kernel of the HPCA algorithm), the PCA, and the History PCA algorithms, while Sec. IV introduces our Structural Health Monitoring testbed, namely a fully operational large-scale installation (90 sensors and two gateways) on a real highway viaduct. Sec. IV-B describes the different hardware architectures employed and Sec. V details the implementation of the algorithm on different platforms. Sec. VI presents the hardware results and the SHM analysis performance after data compression, by including both an operational benchmark (a peak detection algorithm on real-world viaduct vibration data) and an extensive discussion about timing/energy trade-off. Sec. VI-C completes the comparison with a load-reduction trade-off. Sec. VII concludes the paper with final considerations and remarks.

II. Related Works

Large IoT sensor networks that manage significant data flows are getting widespread, leading to high demand for methods and architectures able to continuously gather and process large streams of data. When these streams are collected at a central unit to be stored or processed, the communication or the cost for storage space often represents the system bottleneck [36]. This bottleneck can be solved by data reduction proposed in several real-time systems, by either compressing it or distributing part of the processing throughout the network [37], [38]. Many techniques are also proposed to optimize the

TABLE I
COMPARISON OF DIFFERENT CODING SCHEMES FOR DATA COMPRESSION. ABBREVIATIONS: d, SIGNAL DIMENSION, CR, COMPRESSION RATIO (HIGH: $10-50$, MEDIUM: $5-10$, LOW: $1-5$), EMBED.: LEVEL OF EMBEDDING (GOOD – END-NODE, MEDIUM – GATEWAY, BAD – CLOUD)

Works	Compression technique	Principal domain	CR	Computational cost	Embed.	Dataset
Lossless compression						
Grossberg et al. [30]	LZMA	IoT General Purpose	low	low	good	real
Blalock et al. [31]	LZ4	IoT General Purpose	low	low	good	real
Blalock et al. [31]	Deflate	IoT General Purpose	low	low	good	real
Lossy compression						
Jayawardhana et al. [32]	Compressed sensing	IoT General Purpose	low	$O(d^2/\text{CR})$	good	synthetic
Liu et al. [33]	Wavelet-based	Infrastructure monitoring	high	$O(d)$	medium	synthetic
Alsheikh et al. [34]	Autoencoders (AE)	Temperature and Humidity WSNs	medium	$O(d^2/\text{CR})$	bad	real
Wu et al. [35]	PCA-based	General purpose	high	$O(d^2/\text{CR})$	good	real
This work	PCA-based	General purpose	high	$O(d^2/\text{CR})$	good	real

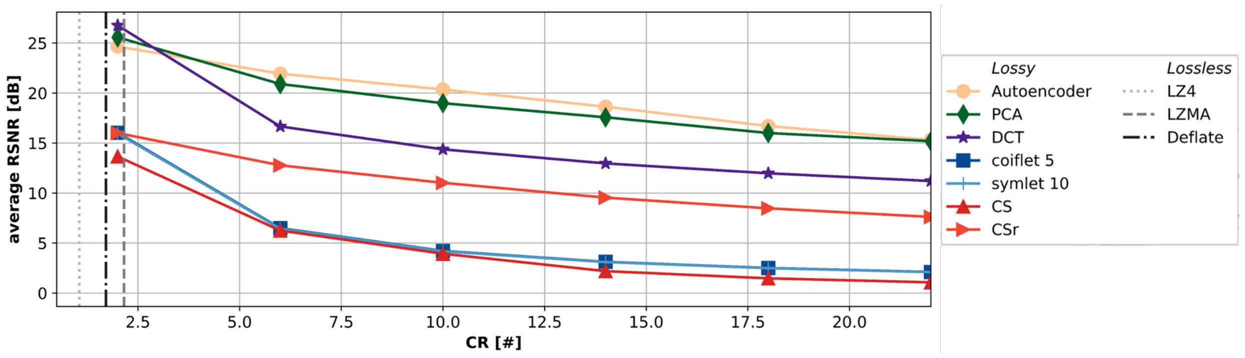

Fig. 2. Comparison of lossy and lossless methods on our SHM vibration dataset. Abbreviations: PCA: Principal Component Analysis, DCT: Discrete Cosine Transform, CS: Compressed Sensing, CSr: rakeness-based.

workload of the network's node, especially in the deep learning field for the production of smart data [39], [40].

This trend has percolated to the SHM field. In [6], a system-level co-design between sensors installation and algorithm resolution has been introduced, which allows a reduction of the data gathered, streamed, and stored if a lower resolution is demanded. Further, moving the processing to the edge, [4] proposes a distributed execution for the eigensystem realization algorithm (ERA), a classical SHM algorithm. The implementation proposed relieves the central unit from the computation and allows to stream only "smart" data, which already contains the diagnosis information. As a drawback, this class of algorithms prevents the system to store the raw recorded data or at least their approximation, which can be useful for additional analysis.

Data compression represents an alternative solution to limit communication bandwidth. Table I summarizes the salient features of a wide range of compression algorithms, divided into two main categories: lossless and lossy.

Lossless methods ensure no loss of information at the cost of a low compression ratio [41], [42]. For instance, LZMA [30], LZ4, and Deflate [31], widely accepted lossless methods used in file compression algorithms, reach average compression ratios lower than $3\times$. On the other hand, lossy methods, which achieve a higher compression ratio [43], leverage the fact that not all the signal information is useful for the analysis.

A well-known lossy method is Compressed Sensing (CS), which allows the implementation of energy-efficient encoders [44], [45]. The upside is that with few linear projections and low computational cost, CS captures the primary information contained in the signal, thus being very suitable for SHM applications [32]. The downside is that the energy efficiency comes at the cost of a lower compression ratio (CR) compared to other methods such as wavelet-based, as shown in [46], [47]. These methods are more power-hungry but still suited for embedded devices.

For instance, [33] combines wavelet transformation with distributed source coding to increase the compression performance further, reaching a compression factor of 50 with synthetic vibration data. However, this high compression factor is mainly due to the very high correlation between the generated synthetic data, which are not representative of other different monitoring scenarios. In fact, in [33], the data are collected from a five-layer civil infrastructure laboratory model, with a distance as small as 15 cm between each layer and a vibration exciter attached to the first one.

A new promising alternative comes from the machine learning world. In [48], it is shown that an autoencoder (AE) can outperform other classical methods such as the ones based on PCA, wavelets, and Discrete Cosine Transform (DCT) with a comparable or lower computational cost. Note that in SHM, auto-encoders have already been successfully employed for temperature and humidity data [34] or embedded in a more complex damage identification system [49]. The main drawback of the usage of autoencoders embedded on edge devices is the need for large training datasets. Moreover, structures often change over time (e.g. due to aging), and are thus non-stationary, forcing the compression algorithms to be re-trained over time. This requires the transmission of a large amount of data for re-training autoencoders in the cloud.

We focus on PCA-based compression [50], which exploits the correlation between signal components to extract the primary information. The compression ratio is similar to the wavelet-based method, as shown in [34], [48], but, similarly to CS, PCA-based encoders require few linear projections to compress the raw signal. As a counterpart, the PCA (similarly to autoencoders) requires to store a considerable amount of data to estimate the principal components needed for the compression accurately. Hence, either the memory footprint or the CPU-time needed for the analysis is often prohibitive for a typical edge or gateway device.

Well known streaming approaches [23], [24], [51]–[57] address this issue. These methods update the estimation of the principal components by considering sequential chunks of data, and thus performing the analysis while the data stream is flowing, without the need of storing it.

This is a field that has gained particular importance in the last few years: however, important contributions date back to 1968 and 1982 when Krasulina's and Oja's methods were presented [51], [52]. They are Stochastic Gradient Descent (SGD) methods applied to minimize two different objective functions whose optimum point is the principal component. Following their intuition, extended versions have been presented in literature to allow the processing of blocks of data as well as the estimation of a subspace with rank higher than one [53], [58].

The aforementioned methods perform a step along the gradient and then orthogonalize the current solution to get a base for the principal subspace. As an alternative, authors in [56] showed that it is possible to follow the gradient along the geodesic in the manifold of orthogonal subspaces, i.e., the Grassmannian manifold. All these SGD-based methods require a fine tuning of the learning rate since low rates slow down the convergence while high rates could cause divergence.

A different approach is the Incremental Singular Value Decomposition (ISVD), presented in [59], which is an exact method to compute the full SVD of a data matrix whose vectors arrive sequentially. However, this method is computationally expensive. To overcome this issue, the authors in [57] propose a modified version that computes the thin SVD whose complexity is of the same order as the SGD-based methods. Moreover, together with a base for the principal subspace, ISVD provides an estimation of the singular values.

History PCA is a novel method presented in [24] whose core resembles SGD-based methods but, similarly to ISVD, it provides an estimation of the eigenvalues and it does not require the tuning of a learning rate. These characteristics make HPCA a suitable method for real world applications. Indeed, the History PCA has been tested on four different large-scale datasets (NIPS and NYTimes from UCI data, and RCV1, KDDB from LIBSVM [24]) achieving lower approximation error in the estimation of the principal components compared to State-of-the-Art methods.

In this paper, we propose to reduce the traffic load in a sensor network for SHM by embedding the HPCA algorithm on either gateways or sensor nodes, maximizing the compression-ratio under the memory constraints characterizing the IoT devices. Our approach leverages the time correlation of the single sensor time series, allowing to treat each sensor node as a unique entity, i.e. a single PCA instance can run independently on each sensor. Moreover, the adoption of a streaming PCA algorithm allows embedding the compression stage directly on the sensor node, reducing the traffic between gateways and cloud as well as between sensors and gateways. To the best of our knowledge, this is the first work that analyzes the embedding of a PCA based compression algorithm at different levels of an SHM net by exploiting the autocorrelation of the signal.

A. *Compression algorithm evaluation*

To validate the effectiveness of the PCA-based compression method in a SHM application based on vibration sensing, we compared its performance with some of the algorithms previously presented. In this manuscript, we consider the PCA as a pure enabler of SHM damage detection algorithms with reduced network traffic (we show an example in Section VI-A). Hence, the assessment is based on our use case dataset and Fig. 2 shows a figure of merits regarding the quality of reconstruction of the signal (in terms of RSNR defined in Section III-D) depending on the compression ratio (CR).

Particularly, we compared PCA with both the lossless and lossy methods. We selected LZ4 [30], LZMA and Deflate methods [31] as lossless methods, while for lossy approaches we considered compression based on Discrete Wavelet Transform (DWT), Discrete Cosine Transform (DCT) [60], Compressed Sensing (CS) [61] and autoencoder [34]. For the DWT implementation, we consider symlet, coiflet, daubechies, and haar families, but in Fig. 2, we only show symlet 10 and coiflet 5 that outperform the others. For the Compressed Sensing approach, along with the standard method [61], we also consider the rakeness method that adapts the encoder stage to the class of signals [62].

We show that the average RSNR obtained with the PCA-based method is comparable with the performance of the autoencoder while outperforming all the other methods for $\mathrm{CR} > 4$. On the other hand, the PCA has a twofold advantage compared to the autoencoder: it requires less data for the training, and the streaming approaches allow to avoid storage of the complete training dataset, which would be prohibitive on an end-node device. It is also noticeable that while considering Table VI the DWT approaches seem to outperform all other methods in [33], they perform poorly when applied to the sensor data considered in this paper.

III. Background & Methods

In this section, we provide a short introduction to the QR-decomposition used in the HPCA algorithm and to the PCA. After, we describe the keys advantages of the HPCA compared to classical PCA and other streaming PCA algorithms, providing an example of the execution of the HPCA on our SHM data.

A. QR-decomposition

Let $\mathbf{A}$ be a real matrix $d \times k$ with $d > k$ and $\mathrm{rank}(\mathbf{A}) = k$, we call QR-decomposition the product

$$\mathbf{A} = \mathbf{QR} \tag{1}$$

with $\mathbf{Q}$ orthonormal matrix ($\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}_k$), $\mathbf{R}$ upper triangular matrix and where $\mathbf{I}_k$ denotes the identity matrix of dimension k. Several methods have been proposed in the literature to compute the QR decomposition, such as Cholesky decomposition, Gram Schmidt, or Householder reduction (HH) [63]. We based our algorithm on HH [64], because, in comparison to the other methods, HH benefits from better numerical stability, and it is suitable for fixed-point quantization [63]. In particular, we are interested only in the computation of the $\mathbf{Q}$ matrix, whereas we discard the $\mathbf{R}$ matrix. Indeed, in HPCA, QR decomposition is only used to ensure the orthonormality among columns and $\mathbf{R}$ is never used (Alg. 1), allowing to save both memory and operations.

Let us consider the QR factorization as the composition of two different phases, QR_1 and QR_2. The former takes as input the matrix $\mathbf{A}$ and computes a set of vectors $\mathbf{w}_i$, with $i = 1, \ldots, k$ and with $\|\mathbf{w}_i\|_2 = \sqrt{2}$, where $\|\cdot\|_p$ stands for l_p-norm. Note that given the iterative process of the Householder reflections on sub-matrixes of input matrix $\mathbf{A}$, the $\mathbf{w}_i$ have progressively lower dimensions, from d to $d - k$. These are the base elements for the Householder matrices $\mathbf{P}_i$ [64]. These vectors are directly stored in the lower triangular part of the input $\mathbf{A}$, and their computation requires both high accuracy and high dynamic range.

Subsequently, QR_2 expands the $\mathbf{w}_i$ with an initial series of 0 to match the dimension d for each vector. The $\mathbf{w}_i$ are used to compute

$$\mathbf{P}_i = \mathbf{I}_d - \mathbf{w}_i \mathbf{w}_i^\top \quad \text{with} \quad \mathbf{Q} = \prod_{i=1}^{k} \mathbf{P}_i \tag{2}$$

requiring only successive matrix multiplications between orthornormal matrices. The range of values for this part of the algorithm can be easily computed: the biggest range is represented by $\mathbf{w}_i \mathbf{w}_i^\top$, which produces values in the range $[-2, 2]$. Conversely, since the $\mathbf{P}_i$ are orthornormal

$$\begin{aligned}\mathbf{P}_i^2 &= (\mathbf{I}_d - \mathbf{w}_i \mathbf{w}_i^\top)(\mathbf{I}_d - \mathbf{w}_i \mathbf{w}_i^\top)^\top \\ &= \mathbf{I}_d - 2\mathbf{w}_i \mathbf{w}_i^\top + \mathbf{w}_i \underbrace{\mathbf{w}_i^\top \mathbf{w}_i}_{\|\mathbf{w}_i\|_2^2 = 2} \mathbf{w}_i^\top = \mathbf{I}_d\end{aligned}$$

and a product of orthornomal matrices is again an orthonormal matrix, all the successive values are bound in the range $[-1, 1]$.

B. Principal Component Analysis

Principal Component Analysis (PCA) aims at reducing the dimensionality of multivariate data while preserving as much of the relevant information as possible. More specifically, PCA refers to the problem of finding the k-dimensional subspace that best approximates in l_2 terms a given dataset, i.e., a collection of signal instances.

Let us define as $\mathbf{X} \in \mathbb{R}^{d \times N}$ a dataset composed by successive input signals $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathbb{R}^d$, and denotes the matrix $\mathbf{V} \in \mathbb{R}^{d \times k}$ whose columns form a basis for the subspace that best approximates the dataset. Dimensionality reduction is achieved by projecting $\mathbf{X}$ on the subspace $\mathbf{V}^\top \mathbf{X} = \mathbf{Y} \in \mathbb{R}^{k \times N}$, thus obtaining a compression ratio $\mathrm{CR} = d/k$ assosiated to the set of reconstructed signals $\tilde{\mathbf{X}} = \mathbf{VY} = \mathbf{VV}^\top \mathbf{X}$.

PCA identifies V as a solution of a minimization problem in which the objective function is the reconstruction error:

$$\mathbf{V} = \underset{\mathbf{V} \in \mathbb{R}^{d \times k}, \mathbf{V}^\top \mathbf{V} = \mathbf{I}_k}{\arg\min} \sum_{j=1}^{N} \left\| \mathbf{x}_j - \mathbf{VV}^\top \mathbf{x}_j \right\|_2^2 \tag{3}$$

To adopt this scheme as a real-time compression algorithm, i.e., $\mathbf{V}$ is used to compress incoming signal instances, data vectors $\mathbf{x}_i$ must be approximate realizations of a stationary stochastic process whose statistic estimation requires a high N value.

The matrix $\mathbf{V}$ can be obtained by means of either the eigenvalues decomposition (EVD) of the sample covariance matrix $\mathbf{\Sigma}_N = \mathbf{XX}^\top / N$ or the singular value decomposition (SVD) of the dataset matrix $\mathbf{X}$, and taking the d-dimensional vectors associated with the k greatest eigen or singular values. Nevertheless, both EVD and SVD require the storage of the entire dataset matrix to perform PCA, making them not suitable for many real cases due to the dimension of $\mathbf{X}$ compared to the resources available on the executing device. For instance, this prevents the embedding of this algorithm on edge computing platforms, which are subject to tight constraints on memory space. This bottleneck can be solved by streaming PCA approaches.

C. History PCA

As a common rule in streaming PCA algorithms, the principal subspace estimation is updated with data samples arriving sequentially without accessing historical data and without the storage of the entire dataset.

Among them, History PCA (HPCA) [65] has recently emerged. Based on the block-stochastic power method [53], HPCA aims at improving its training accuracy by estimating not only a base for the principal subspace but also its corresponding set of eigenvalues. These quantities allow for a better representation of the historical data. The HPCA training procedure is provided in Alg. 1, and it is extensively explained in [24]. The algorithm runs a new step every time a block $\mathbf{X}_\tau \in \mathbb{R}^{d \times B}$ is gathered, consequently updating the compression matrix $\mathbf{Q}_\tau$. After $n = {}^{N}\!/_{B}$ steps the method returns the final estimate $\mathbf{Q}_n$.

The core of the algorithm is contained in line 13, where the estimated principal subspace is updated with the incoming block of data X_τ. Besides, the QR decomposition in line 14 is necessary to ensure the orthonormality among the columns of $\mathbf{Q}$. This task is computed m times, which is proved to increase the accuracy in the principal subspace estimation [24].

Algorithm 1 HPCA

```
1: Input: $\mathbf{X}_1, \ldots, \mathbf{X}_n$, block-size: $B$.
2: $\mathbf{S}_0^{(i)} \sim N(0, \mathbf{I}_d), 1 \le i \le k$
3: $\mathbf{Q}_1 \leftarrow \text{decomposition}_{QR}(\mathbf{S}_0)$
4: for $i \leftarrow 1, \ldots, m$ do
5:     $\mathbf{S}_1 \leftarrow \mathbf{Q}_1 + \frac{1}{B}\mathbf{X}_1\mathbf{X}_1^\intercal\mathbf{Q}_1$
6:     $\mathbf{Q}_1, \cdot \leftarrow \text{decomposition}_{QR}(\mathbf{S}_1)$
7: end for
8: $\lambda_j \leftarrow \|\mathbf{S}_1[:, j]\|_2$ for $j = 1, \ldots, k$
9: $\mathbf{\Lambda}_1 \leftarrow diag(\lambda_1, \ldots, \lambda_k)$
10: for $\tau \leftarrow 2, \ldots, n$ do
11:     $\mathbf{Q}_\tau \leftarrow \mathbf{Q}_{\tau-1}$
12:     for $i \leftarrow 1, \ldots, m$ do
13:         $\mathbf{S}_\tau \leftarrow \frac{\tau-1}{\tau}\mathbf{Q}_{\tau-1}\mathbf{\Lambda}_{\tau-1}\mathbf{Q}_{\tau-1}^\intercal\mathbf{Q}_\tau + \frac{1}{\tau}\frac{1}{B}\mathbf{X}_\tau\mathbf{X}_\tau^\intercal\mathbf{Q}_\tau$
14:         $\mathbf{Q}_\tau, \cdot \leftarrow \text{decomposition}_{QR}(\mathbf{S}_\tau)$
15:     end for
16:     $\lambda_j \leftarrow \|\mathbf{S}_\tau[:, j]\|_2$ for $j = 1, \ldots, k$
17:     $\mathbf{\Lambda}_\tau \leftarrow diag(\lambda_1, \ldots, \lambda_k)$
18: end for
19: Output: $\mathbf{Q}_n$
```

HPCA algorithm starts from an initial matrix $\mathbf{Q}_1$ that comes from a random initialization [24] (see lines 2–3). Nevertheless, warm start procedures can be adopted to reduce the time of convergence, e.g., warm starts procedures used in [55], [58] for other streaming PCA methods. A possible warm start, also working for HPCA, consists of a matrix $\mathbf{Q}_1$ obtained as the left-singular vector from the rank-k thin SVD computed on an initial data block. Preliminary results show a negligible difference in terms of the time of HPCA convergence such that we maintain a random initialization. We will further explore this aspect during our future work.

The computational cost of the HPCA is dominated by the matrix multiplications $O(dk(k+B))$ and by the QR decomposition $O(dk^2)$. The memory occupancy is $\sim O(d(k+B))$, implying a gain of $^N/_{(B+k)}$ compared to the classical PCA. With $B = 1$, the algorithm reaches its lowest memory footprint.

These values are consistent with other streaming PCA algorithm known in literature [52], [53], [55], which present complexity $O(dk^2)$ and $O(dk)$ memory. The advantages of HPCA reside in the robustness of the parameter tuning, as shown in the next paragraph and the improved rate of convergence granted by the iteration of the internal loop, as demonstrated in [65].

Note that the training dataset in our use case is derived from a single 1-D acceleration time series, by separating the recorded signal in d-dimension time-windows. This segmentation procedure allows us to consider each window as an input sample for the algorithm.

D. Parameter Tuning

For the parameter tuning, we use a dataset comprising a single-axis time series gathered by one sensor node of the installation. We consider two traces, each one 1 week long. Since the signal is affected by temperature and traffic intensity, the 1-week period takes into account the daily periodicity and the different traffic conditions of the weekdays. The former trace is used to estimate the compression matrix $\mathbf{Q}$ while the latter is needed to assess the quality of service (QoS), measured as the reconstruction signal to noise ratio RSNR :

$$\text{RSNR} = 20\log_{10}\left(\frac{\|\mathbf{x}\|_2}{\|\mathbf{x}-\hat{\mathbf{x}}\|_2}\right)$$

where $\mathbf{x}$ represents a generic signal instance and $\hat{\mathbf{x}}$ is the correspondent reconstruction after PCA-based compression.

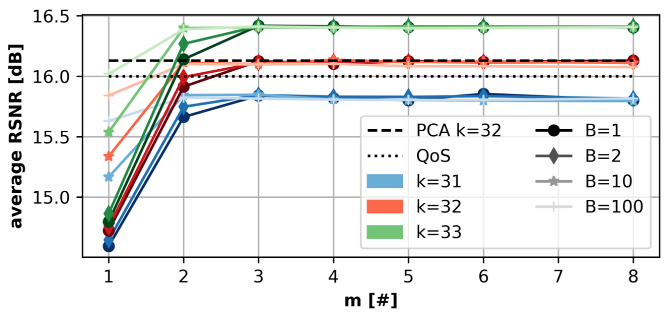

Fig. 3. HPCA performance in terms of RSNR depending on the number of internal loops m for different block-size b and number of principal components k.

The parameters that are intrinsic in PCA-based compression are d, (i.e. the signal dimension) and k (i.e. the dimension of the compressed signal). Although in static data compression, d is a given parameter, temporal data allows for sweeping the values of d. It is noteworthy that keeping constant the Quality of Service (QoS) results in a compression factor monotonically increasing with d. Hence, for the testbed presented in Section IV-A, as a trade-off between the compression factor and real-time processing, we upper-bound the parameter d to obtain a maximum delay of t_w = 5s (compliant with the other near real-time algorithms running in the testbed), that results in a maximum $d \le 500$ (since the dataset is sampled at 100 Hz). Therefore we obtain $N \sim 120000$ signal instances for the training and validation 1-week sets.

The parameter k is determined by finding the minimum number of principal components that, when used to compress a validation set, satisfies the required quality of service (QoS). To minimize error rate in peak detection algorithm presented in section VI-A, we consider RSNR $= 16$ dB. On the validation set, classical PCA analysis reaches average RSNR $= 16.13$ dB with $k = 32$ components, i.e., obtaining a $CR = 15.6$. A minimum RSNR of 16 dB is used throughout the rest of the paper as a minimum constraint for the quality of reconstruction. We do not consider it as an objective, but as a constraint to be compliant with. In other words, we search for a set of solutions that respect this constraint while presenting possible trade-offs in terms of energy, delay, and compression ratio.

The HPCA algorithm relies on two further parameters, B, the input block-size, and m, the number of iteration for the internal loop. Hence, B mostly affects the memory footprint that each step of the algorithm requires, while m influences the computational cost. Fig. 3 shows the average RSNR depending

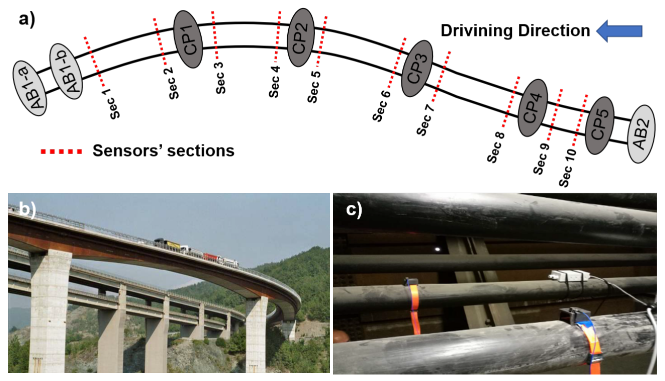

Fig. 4. Monitoring system installation: (a) Plan view of the monitored highway viaduct. (b) Overview of the monitored highway bridge. (c) Sensor installed on the external steel tendons. The sensor is based on the STMicroelectronics STM32F405RG and features an LIS344ALH as an analog accelerometer.

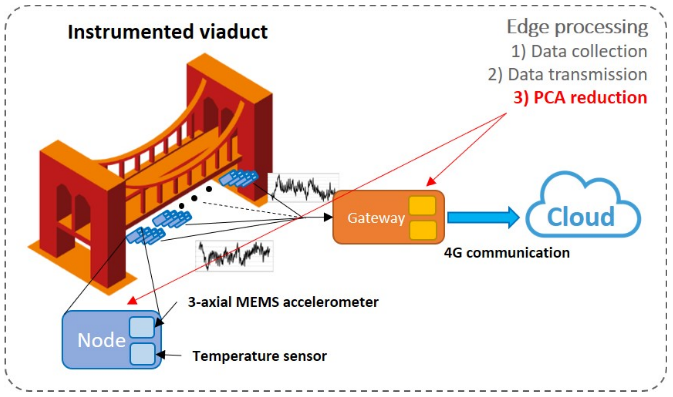

Fig. 5. Monitoring system installation description with the two possible PCA reductions location.

TABLE II
STRUCTURAL HEALTH MONITORING SYSTEM OF THE DESCRIBED BRIDGE. THE NODES COMPRISING THE TWO SENSORS (SEC. IV-B2) ARE DISTRIBUTED OVER THE 10 SECTIONS IN GROUPS OF 9 ± 3.

	#	Link	Memory	Computation
Nodes	90	CAN bus	192 kB	1-core, 168 MHz
Gateways	2	CAN bus/4G	1 GB	4-cores, 1.2 GHz
Cloud	n.a.	4G	∞	∞

on m, for different values of B and number of projections k. In all cases, performance saturates with $m = 3$ independently on the adopted block size B. As expected, the higher k, the higher the average RSNR , demonstrating that HPCA exhibits good robustness to changes in parameters.

With $k = 32$ (same of the PCA), $m = 3$ and $B = 1$ the average RSNR measured on the validation set is $16.11\,\mathrm{dB}$, just $0.02\,\mathrm{dB}$ below the value reached with traditional PCA and still above the value of QoS required, i.e. $\mathrm{RSNR} = 16\,\mathrm{dB}$.

Note that the parameters m and B are strictly related. Performing iterations of the update step has a similar effect to a single update with a larger step size, i.e., both approaches refine the gradient direction of the current step. This behavior is confirmed in Fig.3, where for low values of m, larger block sizes B provide a significant increase in RSNR.

IV. STRUCTURAL HEALTH MONITORING INSTALLATION

Our real-life SHM testbed is a highway bridge located in Italy on which many maintenance interventions have already been undertaken from its opening (opened to traffic in 2006). The monitored structure is presented in Fig. 4: Fig. 4a introduces a planar model of the viaduct, with the sections of the monitored tendons highlighted, Fig. 4b shows an aerial image of the bridge, and Fig. 4c portraits the MEMS installed on tendons, which sense the bridge vibration.

A. Monitoring System

The installed infrastructure on the bridge is a vibration-based SHM system, which exploits the vibration of the bridge under traffic condition to detect possible degradation or damage of the monitored structure. The installation, depicted in Fig. 5, is divided into two main blocks: the in-situ node with the gateways supervised network, and the cloud part. A summary of the system elements is reported in Table II.

The in-situ section is the most critical part since several problems arise from the interconnection of the different elements. In the current setting, the sensor nodes are 90, and they are placed on the pre-stressed tendons, at most two sensors per tendon. Each node, presented in [66], is equipped with a micro-controller and a MEMS triaxial accelerometer, which measures the acceleration in three orthogonal directions (x, y, z), with an angle between each two of those directions of $90^\circ \pm 2\%$. With the goal of high acceleration resolution and prevention of aliasing, the acceleration is sampled at $25.6\,\mathrm{kHz}$ by the internal ADC and then filtered and decimated with a 6-state FIR filter which reduces 256 samples to a single value, thus implying a $100\,\mathrm{Hz}$ final sample frequency.

The quality of the produced acceleration data has been assessed through an extended preliminary experimental phase [67], [68] that has verified that this sensing system allows for accurate estimation of the natural frequencies, often taken in literature as an index of the integrity of the monitored object [20].

Once the data is filtered, the sensor node transmits it to a local gateway through CAN-bus. The system comprises two gateways, each one connected to 45 sensors. The bit rate R_b on this connection is

$$R_b = N_S \times N_{\mathrm{ax}} \times f_s \times L_s$$

where N_S is the number of sensors, $N_{\mathrm{ax}} = 3$ is the number of axes of each accelerometer, $f_s = 100\,\mathrm{Hz}$ the sampling frequency and $L_s = 16$ is the bitwidth of each sample in bit. Since in the considered installation the CAN-bus requires $R_b \leq 250\,\mathrm{kbps}$, each gateway can manage maximum ~ 50 sensors. As expected, the link data rate is the limiting factor for the size of sensor clusters managed by a gateway.

The data collected by the gateways are sent via Ethernet to the "Ubiquity Nano M5" station located halfway between the viaduct ends. M5 station is also connected via 5 GHz point-to-point Wi-Fi to the access point, which transfers the whole data to the cloud.

The cloud system is composed of a storage platform and a computing machine allocated on the IBM cloud service: we use the IBM Cloud Object Storage for saving data as parquet files and we allocate a machine with 2 nodes, 4 cores, and 16 GB RAM per node. The acceleration, along with the temperature and humidity data, is stored in a cloud monitoring infrastructure, which allows real-time access and analysis. The data is processed to detect unusual patterns that do not fit the normal behavior of the structure, such as abrupt damages or progressive degradation.

B. Hardware platforms

1) Gateway: In the current installation the IoT-gateways are composed by **Raspberry Pi 3** module B [29] (RPi3), but in this paper we also consider the Samsung **ARTIK 710** Module [28] as an alternative.

The RPi3 is a single-board computer initially developed for teaching applications. Now, it is actively used in many fields, such as robotics, smart sensor control, and structural health monitoring. The board comprises the Broadcom BCM2837 SoC, equipped with a 1.2 GHz 64-bit 4-core Cortex-A53, and 1 GB low power DDR2 clocked at 900 MHz.

The ARTIK 710 Module is an embedded computing System-in-Module by Samsung targeted for high-end gateways with local processing and analytics. It consists of an 8-core 64-bit ARM Cortex-A53 running at 1.4 GHz with 256 kB shared L2-Cache, and two 512 MB DDR3 16-bit memory chips with 32-bit memory interface, which provides a throughput of 6.4 GB/s.

2) Sensor-Node: The sensor-node is based on the STmicroelectronics **STM32F405RG** and features the LIS344ALH [69] as analog accelerometer, along with the humidity and temperature sensor HTS221 [70]. The STM32F405xx family of MCU is prevalent for embedded computing, due to the relatively high operating frequency and the floating point (FPU) unit with a full set of DSP instruction. Our STM32F405RG unit contains an ARM 32-bit Cortex-M4 CPU with FPU running at 168 MHz, 192 kB of SRAM, and 1 MB of Flash memory. The board also includes the CAN bus communication to manage the nodes-gateway communication.

V. HPCA Implementation

Here, we provide an overview of the implementation and optimization of the HPCA on the presented hardware architectures. In particular, we focus on the two key constraints of these platforms, namely the CPU-time t_{CPU} and the memory M needed to execute the HPCA algorithm for a single time series. To ensure real-time operation $t_{\text{CPU}} < t_{\text{w}} = {}^{d}/_{f_s} = 5\,\text{s}$. Gateway platforms must run the algorithm for $N_S = 45$ 3-axial sensors. The main challenge is represented by the time constraint as the algorithm must run for every axis of every sensor, $t_{\text{CPU}} * N_S * N_{ax} < t_{\text{w}}$. In contrast, when considering the option to run HPCA on sensors, it needs only 3 iterations (one for each acceleration axis, $t_{\text{CPU}} * N_{ax} < t_{\text{w}}$), but it has to deal with a strict memory constraint, i.e., $M < 192\,\text{kB}$.

To handle these challenges, we compare four different implementations, two running on gateway (GT) and two running on sensors (SNS):

GT1 is a 4/8-core version: a single iteration of the HPCA is parallelized on each core available;
GT2 is a 4/8-core sensor-level parallelized version: for each free core, one iteration of the HPCA runs on it, separately;
SNS1 is a 1-core floating point version;
SNS2 is a 1-core fixed point (16 bit) version.

A. Gateway implementations

The two versions of the History PCA proposed for the gateways are implemented using optimized Numpy Python 3.5 library [71], relying on highly optimized BLAS and LAPACK libraries for linear algebra computation. The focus of the two implementations is the reduction of the execution time of the HPCA.

GT1 is characterized by the internal parallelization of the HPCA algorithm, such that all the available cores are used to execute a single step of the HPCA algorithm and for a single trace. As a result, **GT1** has a minimal memory footprint (gateway process data from only one sensor at a time). However, computational time does not scale with the number of core n_C since many operations (e.g. QR decomposition) can not be fully parallelized.

To cope with this limit, we introduce **GT2**, which processes data from multiple sensors in the same time slot. GT2 is based on a single-core version of the HPCA iteration, which is executed n_C times in parallel on n_C different input time series coming from n_C different sensors. Therefore, GT2 achieves a near-ideal speed-up, which is only limited by the simultaneous accesses to the memory. As a counterpart, since n_C instances of the HPCA are parallelly run, GT2 requires $n_C\times$ higher memory footprint.

B. Sensor-node implementations

Both proposed HPCA implementation running on sensor nodes exploits optimized C code, with CMSIS matrix operations to minimize the execution time.

SNS1 implement the HPCA algorithm in floating-point precision without any parallelized task. The great disadvantage of SNS1 is related to memory occupation. A straightforward deployment of the proposed method could not match the actual memory constraint of the sensor-nodes, and it could limit the maximum instance length d.

SNS2 aims at solving the memory issue by means of a 16-bit fixed-point implementation. With this setting and by exploiting the CMSIS highly optimized matrix operations, **SNS2** reduces both the time and the footprint by a factor of two, storing 16 b instead of 32 b for each element, and using the SIMD MACs for matrix multiplications. These advantages come at the cost of a decrease in reconstruction accuracy. To cope with it, careful tuning of the data quantization has been investigated. We address the two critical parts of the Algorithm 1 with two different operating ranges: computation in line 13 represents the former (mainly matrix multiplications) while the QR decomposition in line 14 is the latter. For the first task, we use a high number of integer bits $q_1 = 5\,\mathrm{b}$ to capture the high dynamic range of the input signal[1].

In the QR decomposition, as anticipated in Section III-A, we split this task in two stages where the number of integer bits is $q_2 = 4\,\mathrm{b}$ for QR_1 (computation of the vectors $\mathbf{w}_i$) and it is $q_3 = 2\,\mathrm{b}$ in the QR_2 stage (multiplication of the matrices $\mathbf{P}_i$). All reported number of integer bits are the result of a trade-off between dynamic range and precision assessed. Indeed, q_3 has been analytically derived, while q_2, as well as q_1 in the first step, has been computed through a grid search experiment on a cross-validation set composed by 4 hours of recording. We keep floating-point representation for the normalization of the $\mathbf{w}_i$ vectors ($\|\mathbf{w}_i\|_2 = \sqrt{2}$, see Section III-A), since the memory occupancy is not significant while more accurate values strongly impacts the performance.

As a result, the average reconstruction quality for SNS2 is only slightly lower compared to the one observed with SNS1. To compensate for this residual performance loss, we lower the CR with an increase of the parameter k (from 32 to 50) such that the target quality of service is maintained (average $\mathrm{RSNR} = 16.02\,\mathrm{dB}$). Note that we do not account for a 8 b fixed-point implementation because, in that case, the approximation error with any combination of the parameters k, m, B is such that the algorithm fails in reaching the minimum acceptable RSNR .

VI. Experimental Results

We define forced vibration as the vibration that results from the application of an external time-dependent disturbance. Note that under a forced vibration, a structure tends to vibrate at its natural frequencies. Even a subtle variation can be a symptom of structural damage or deterioration [72], and it is paramount to observe how the structure oscillates.

In the next paragraphs, we first show how a peak-tracking algorithm maintains its accuracy by working on the reconstructed signal instead of the original one. We use this algorithm to demonstrate the suitability of a PCA-based approach to reduce the traffic on a SHM sensor-network while maintaining the accuracy of the analysis performed on the data. Noteworthy, the embedding of a PCA solution on the very edge of the network can further enhance a SHM system, for example, with an additional standalone anomaly detection solution, as demonstrated in [27].

We tested the HPCA compression in both its floating-point (GT1, GT2, SNS1) and fixed-point (SNS2) implementation in the viaduct health monitoring scenario. We provide a detailed analysis of the energy and delay of the HPCA implementation on both the levels of our SHM network, namely the gateway and the sensor-node, discussing the optimal implementation and the optimal hardware for the application.

Finally, we discuss the trade-off in load reduction at different levels of the networks by analyzing the difference between sensor and gateway deployment. If not differently specified, all the implementations achieve a level of reconstruction $\mathrm{RSNR} > 16\,\mathrm{dB}$.

[1]input instances were normalized by adopting an average and a standard deviation previously estimated on a training set.

A. Peak Detection

We first introduce an approach for peak detection, used in our SHM system to track the natural frequencies of the structure and, eventually, damages in the tendons of the viaduct. Note that this technique is applied to each sensor data stream separately, thus resulting in parallel tracking of multiple structural elements. Moreover, since the tendons are redundant, a damage on a tendon is not catastrophic for the entire viaduct. Hence, an anomaly detection of a tendon, even if late, serves as a precursor of a more extensive problem on the whole viaduct, and it is used to trigger the maintenance intervention. In Sec. VI-A1 we show an example of a tendon breakage happened on the structure. The natural oscillating frequencies can be extracted from the sensor data by detecting the peaks in the signal spectrum. Although the number of peaks and their position depends on the length and on the strain at which the tendons are pre-stressed, the example in Fig. 6 well represents a general case. In fact, most of the signals manifest pairs of peaks nearly evenly distributed in the frequency domain.

The Power Spectral Density (PSD) is estimated by averaging the periodograms over 18 non-overlapping Hanning windows of 200 s each. Hence, every hour, an estimate is produced with a frequency resolution of 5 mHz, which is needed to detect small relative variations of peak frequencies. The resulting profiles are smoothed by a Savitzky-Golay filter (length 11, degree 3) and processed by a peak-picking method. The peak-picking method extracts the highest 15 local maxima that have prominence[2] lower than half of their height. Finally, peaks with a maximum distance of 0.2 Hz and belonging to successive time frame are grouped.

Fig. 6 shows an example of spectrum estimation from original and reconstructed signals of the x-axis vibration signal from one of the viaduct tendons. We consider both the floating-point and 16 bit fixed-point implementation of HPCA compared to the classical PCA method with the set of parameters that guarantees the target RSNR . All four spectra share the same profile in the region near the peaks, but they significantly differ in the other bands. Both PCA and floating-point HPCA show a filtering effect, while the profile of fixed point HPCA remains at the same level as the original one but with a more noisy trend.

The performance of the peak-tracking algorithm is tested on an additional 1-week test set following the two 1-week periods used as training and validation sets. The figure of merit used to validate the approach is the error in the peaks detection made on reconstructed signals compared to the case where the original signal is considered. The error regards both frequency (err_f) and amplitude (err_A) of the peaks and it is computed in terms of difference between the uncompressed and reconstructed cases: $\mathrm{err}_f = \left|f^{\mathrm{ref}} - f\right|$, $\mathrm{err}_A = \left|A^{\mathrm{ref}}_{dB} - A_{dB}\right|$, where the pair $(f^{\mathrm{ref}}, A^{\mathrm{ref}}_{dB})$ represents frequency and amplitude in the uncompressed case, and (f, A_{dB}) the peak characteristic in the case of interest.

[2]In topography, prominence is a measure of the independence of a peak and is computed as the height of a peak relative to the lowest contour line encircling it but containing no higher peaks.

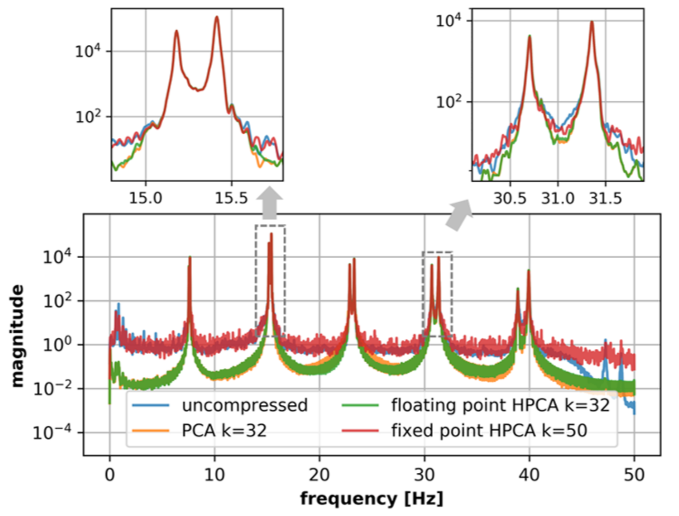

Fig. 6. PSD of the x-axis acceleration signal of one of the sensors in the viaduct monitoring system. First ten peaks are used to monitor the health condition of the viaduct.

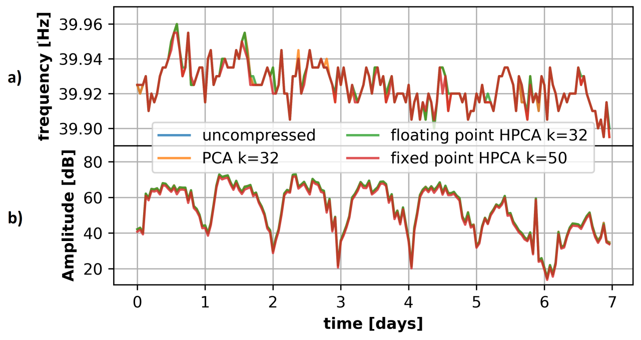

Fig. 7. Peak tracking of the $\sim 39.9\,\mathrm{Hz}$ peak over a period of 1 week, using original signal and reconstructed signal from PCA, floating point HPCA and fixed point HPCA. Panel **a** shows the frequency tracking, while panel **b** the amplitude one.

Tab. III summarizes the results in terms of mean, standard deviation (std), and max value observed over the ten peaks and the whole test set. Both in frequency and amplitude, all the investigated approaches obtain a low mean error. It is worth noting that the max err_f recorded in the test set is just 4-5 times the PSD frequency resolution which is lower than the resolution needed to detect physically meaningful frequency shifts.

To better appreciate performance, Fig. 7 depicts the frequency and amplitude profile of the $\sim 39.9\,\mathrm{Hz}$ peak over the 1-week test set period for the different approaches along with

TABLE III
PEAK DETECTION PERFORMANCE OVER 1 WEEK. MEAN = ERROR MEAN, STD = STANDARD DEVIATION OF THE ERROR, MAX = MAXIMUM ERROR.

	frequency [mHZ]			amplitude [dB]		
	mean	std	max	mean	std	max
PCA	0.21	1.17	20	0.03	0.07	0.71
floating-point HPCA	0.21	1.18	20	0.03	0.06	0.80
fixed-point HPCA	0.41	1.78	25	0.52	0.50	2.27

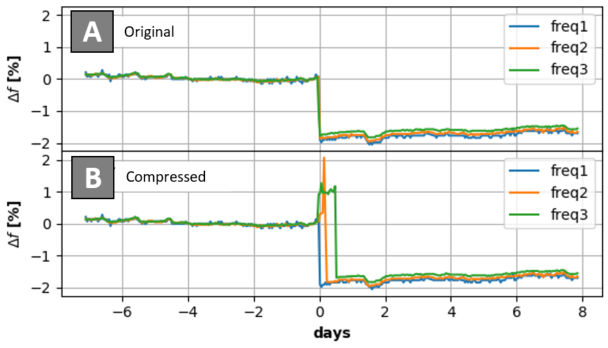

Fig. 8. Shift in the natural frequencies of the viaduct after a tendon strand break. In the upper part, the shift observed using the original signal, in the lower part the one reconstructed using HPCA.

the no compression case. Although it is the peak with the highest err_f and err_A, the difference between the curves is negligible.

1) Damage Identification: We tested this algorithm to benchmark its capability of identifying damages in the viaduct structure. In particular, during the monitoring period, the viaduct experimented a tendon strand breakage, triggering a maintenance intervention to check the conditions of the whole structure. We tracked the natural frequencies before and after this event: Fig. 8 shows the change from the initial natural frequencies; part A depicts the variation computed with the usage of the original signal, part B of the reconstructed one. The natural frequencies are computed using the x-axis and the tracking algorithm presented in the previous section. We used the floating-point HPCA with $k = 32$ to compress the signal in part B. After the break, we observed a 2.0% lowering in the natural frequencies. Noteworthy, using both the original and the compressed signal, this shift can be observed with the same accuracy, with a negligible loss on the average shift accuracy (less than 0.1%).

B. Platforms and implementations: Energy & time trade-off

We use the energy consumption and the execution time to highlight the difference among implementations and hardware platforms, identifying optimal choices for our SHM sensor-network. At the same level of the network (i.e. gateway and sensors), we selected the best performing solution in terms of energy, that respects the near real-time constraint imposed by the gathering of the sensor data (fs = $100\,\mathrm{Hz}$). Note that minimizing the energy entails several advantages on edge devices, such as *i)* an higher number of sensors supplied by the same power bus, *ii)* increasing the number of tasks executed by a single node with the same energy budget, and *iii)* allowing for an higher battery life of a possible future battery powered sensor network. First, we compare the two gateway algorithms and the two gateway platforms, and then we show the differences between the two sensor-node versions.

To measure execution time, speed-up due to parallelization, and energy consumption of the HPCA iteration (the τ-loop in Alg. 1), we consider a full pass through the whole training set. We use the default power mode for all our experiments on

TABLE IV
PERFORMANCE OF GT1 VS. GT2 ALGORITHMS ON RASPBERRY PI 3. RESULTS REFER TO THE EXECUTION WITH $d = 500$, $B = 1$, $k = 32$, $m = 3$. IN BRACKETS THERE ARE SPEED-UP AND ENERGY SAVING WITH RESPECT TO THE 1-CORE EXECUTION.

	CORES [#]	1	2	4
GT1	time [ms]	44.6 (1×)	38.4 (1.2×)	35.3 (1.3×)
	energy [mJ]	18.8 (1×)	26.9 (0.7×)	44.0 (0.4×)
GT2	time [ms]	44.6 (1×)	24.1 (1.9×)	13.3 (3.4×)
	energy [mJ]	18.8 (1×)	16.8 (1.11×)	16.5 (1.13×)

TABLE V
PERFORMANCE OF GT2 ON ARTIK 710 VERSUS RASPBERRY PI 3. RESULTS REFER TO THE EXECUTION WITH $d = 500$, $B = 1$, $k = 32$, $m = 3$. MM AND QR STAND FOR MATRIX MULTIPLICATION AND QR DECOMPOSITION. IN BRACKETS THERE IS SPEED-UP COMPARED TO THE 1-CORE EXECUTION.

n_C	1	2	4	8
Samsung ARTIK 710				
time [ms]	34.7 (1×)	17.7 (2.0×)	9.0 (3.9×)	4.9 (7.1×)
MM [ms]	11.1 (1×)	5.7 (1.9×)	3.0 (3.7×)	1.6 (6.9×)
QR [ms]	22.6 (1×)	11.5 (2.0×)	5.9 (3.9×)	3.1 (7.2×)
Rasberry Pi 3 Model B				
time [ms]	44.6 (1×)	24.1 (1.9×)	13.3 (3.4×)	n.a.
MM [ms]	16.2 (1×)	9.3 (1.8×)	5.1 (3.2×)	n.a.
QR [ms]	27.0 (1×)	14.1 (1.9×)	7.8 (3.5×)	n.a.

the Rpi 3 and the Artik 710 Module, and an external Keithley 2400 SourceMeter SMU for power measurements.

Nevertheless, minimizing the energy budget on a Wireless Sensor Network or, in general, on an edge computing platform, entails several advantages. For instance, it is possible to deploy a higher number of sensors on the same power bus, without increasing the maximum output current. Furthermore, a lower energy budged allows to increase the number of tasks that are executable on the edge node, simplifying also the deployment of a future battery powered version of the network.

1) GT1 vs. GT2 on Rpi3: Here, we analyze the execution time and the energy required by the RPi3 gateway to run GT1 and GT2 implementations with an increasing number of cores. Since we want to evaluate the effects of parallelization, the execution time t_{ex} refers to the inverse of the throughput, the number of results produced per unit of time. For GT1, since the parallelization is internal, t_{ex} is equivalent to the latency of a single HPCA instance. Differently, in GT2, t_{ex} is obtained by running a HPCA instance for each one of the n_C considered cores and dividing the total time by n_C. The results are reported in Table IV. All the experiments are conducted considering the set of parameters obtained as result of the tuning procedure (Sec. III-D), i.e., $d = 500$, $k = 32$, $B = 1$, $m = 3$.

With this setting, a HPCA iteration requires 44.6 ms to run on a single-core. The parallelization on 4 cores results in a 1.3× and 3.4× speed-up for GT1 and GT2, respectively. The low speed-up of GT1 is due to the small dimension of the input data that leads to very high overhead. Instead, the not ideal speed-up of GT2 ($3.4 < n_C = 4$) is probably caused by the high number of simultaneous accesses to the DDR2 memory. That is shown in the lower part of Table V that analyzes the speed-up of GT2 on RPi3. Note that with the single-core configuration the time to process the 3-axial signals from all 45 sensors managed by a gateway ($t_{\text{CPU}} = N_S N_{\text{ax}} t_{\text{ex}} = 6.02\,\text{s}$) does not satisfy the constraint of $t_{\text{CPU}} < t_{\text{w}} = 5\,\text{s}$. Parallelization is, therefore, essential on RPi3 to guarantee the feasibility.

Not only GT2 is faster than GT1 but is also more energy-efficient, showing a 1.13× energy saving of the multi-core execution compared to the single-core and a 2.7× saving compared to its GT1 counterpart.

In the rest of the section, we will benchmark the results using GT2 as best performing implementation in terms of execution time and energy saving on the gateway.

2) ARTIK710 vs Rpi3: We here assess the ARTIK710 as an alternative gateway to the Rpi3 to improve the performance of the current installation. To evaluate the differences, we benchmark the two platforms using single-core and maximum core executions. Table V portraits a detailed comparison of the HPCA execution on the two platforms. Using a single-core configuration, the ARTIK710 and the RPi3 have a comparable execution time (34.7 ms vs. 44.6 ms). Nevertheless, by considering the best performance configuration that exploits all available cores, the ARTIK710 is 2.7× faster. The higher speed-up of this platform is mainly due to the higher number of cores but also given by the faster DDR3. Indeed, with the same number of cores, Artik710 achieves higher speed-up (3.9× vs. 3.4×).

Note that, considering their most performing HPCA implementation, both Artik710 and RPi3 platforms allows a single gateway to process all the 90 sensors of the considered installation. The system bottleneck, therefore, remains the sensors-gateway communication that limits to $\sim$ 50 the number of sensors per gateway.

We also evaluate the relation between energy and the size of the input block B of the HPCA iteration. Fig. 9 shows the trade-off between memory occupancy and energy consumption. Note that, the memory occupancy of the HPCA algorithm is represented by

$$M = (3dk + dB + k^2)d_s$$

where dB accounts for the input $\mathbf{X}_\tau$, $3dk$ and k^2 for the intermediate data to compute the $\mathbf{Q}_\tau$ matrix, and d_s is the size of the data (e.g. 4 B for a float). For high values of B (i.e., $B \gg k$), the memory occupancy is dominated by X_τ, while for $B \ll k$, it is determined by the matrices S and Q_τ.

Increasing the value of B reduces the number of $\mathbf{X}_\tau$ blocks $n = N/B$ that compose the training set, and consequently, the number of iteration required by the HPCA to obtain the result. For instance, by increasing B from 1 to 50, the HPCA necessitates 1.5× memory (from 198 kB to 296 kB and saves almost 50× energy consumption on both the platforms. Overall, setting $B = 1$ (minimum memory footprint) allows a 1221× memory reduction compared to standard PCA, moving from 242 MB to 198 kB which results in passing from 65.3 GB to 53.5 MB if we consider the whole 90 sensors installation.

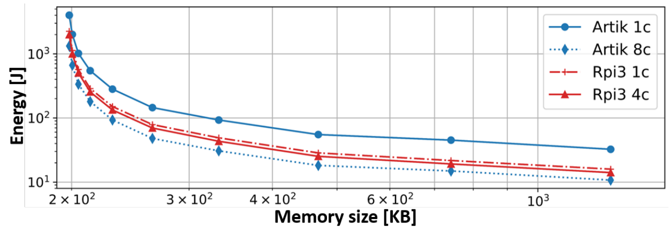

Fig. 9. Trade-off between energy consumption and memory occupation after a full pass over all the training data of the HPCA. Settings: $d = 500$, $k = 32$, $B \in [1, 512]$, $m = 3$. c = core.

Fig. 9 also depicts the different energy consumption between the two platforms. On a single-core, HPCA is computed bound, and the Artik module shows higher energy consumption due to the DDR3 RAM, which is more power-hungry than the DDR2 mounted on RPi3. When we move to the multi-core execution, the bottleneck becomes the memory access. The DDR3 in the Artik module grants for quicker access to the RAM that allows the eight cores to run more efficiently, resulting in lower power consumption compared to the single-core configuration. Conversely, the DDR2 mounted on RPi3 does not allow the four cores to exploit the maximum speed-up since most of the time is spent in load-store operations, and the speed-up is not able to compensate the higher power consumption.

3) SNS1 vs. SNS2: In this paragraph, we move to the sensor node to compare the **SNS1** and **SNS2** implementations. Since the mounted uC is characterized by a strict memory constraint (192 kB), we consider the lowest memory configuration, with $B = 1$, $k = 32$, and $m = 3$. Even with this setting, SNS1 does not fit the uC memory; to overcome this issue, we developed the 16 b fixed-point implementation that ideally halves the memory occupancy. In practice, the memory is reduced only of 22% (from 200 kB to 156 kB) as the fixed point requires an increase of the k (from 32 to 50) parameter to meet the QoS required by the application. Indeed, this reduction is sufficient to meet the memory constraints of a microcontroller, and hence we consider this implementation in the gateway-sensor comparison.

We test SNS1 and SNS2 with a common set up, i.e. by decreasing both d and k to 100 and 7 (we kept constant $CR \sim 15$). Note that this comparison is only synthetic, since using the small $k = 7$ with the reduced $d = 100$ does not allow to maintain the minimum RSNR. Thus, it can not be considered as a possible solution for the sensor-deployment. We use this experiment to provide a detailed comparison of SNS1 and SNS2, highlighting the energy benefit of using SNS2. As expected, the SNS2 (fixed-point) shows a 2.2× faster execution and consumes 2.2× lower energy when compared to the SNS1 floating-point implementation (STM32F405RG column of Table VI). The lower execution time is achieved thanks to the DSP SIMD instructions.

Hence, the fixed-point implementation not only makes the HPCA runnable on the uC but also outperforms SNS1 in terms of execution time and energy consumption. As a counterpart, the SNS2 reaches a lower compression factor, thus implying that SNS1 could be a better choice for sensor-nodes with higher on-board memory.

C. Gateway vs. Sensor Node: load reduction trade-off

We finally compare the two implementation paradigms, namely the gateway implementation and the sensor-node one, by giving a short explanation of the most crucial trade-off, i.e. load reduction at different sensor-network levels. We analyze the system while moving the workload to the edge, from GT2 running on RPi3 to SNS2 running on STM32F405RG with $d = 500$, $k = 32, 50$, $m = 3$, and $B = 1$ as parameters.

First, moving the HPCA to the sensors allows managing bigger networks, increasing the **scalability** of the sensor-network without having limits of processing time on the gateway. Therefore, a new possible installation could be composed of a much higher number of sensors with a single gateway that manages the signal from all the nodes. It is noteworthy that it is not necessary to minimize the delay at this level, but simply to respect the real-time constraint imposed by the gathering of the sensor data. For the fixed $d = 500$ and $k = 50$ (imposed by the application to obtain $\mathrm{RSNR} > 16\,\mathrm{dB}$), the time to store a new batch of data is 5 seconds (sample frequency of $100\,\mathrm{Hz}$) and the time of PCA processing on the STM32F4 is $569\,\mathrm{ms}$. Consequently, the only limitation becomes the connection' bandwidth between the sensor and the gateway, as explained in Sec. IV-A. This limitation only depends on the communication protocol of the network, such as wireless networks or CAN connection (as in our case).

Nevertheless, the **network traffic** is the most impacted metric. Deploying the HPCA on the sensor causes a reduced compression of $CR = 10$ to maintain the same quality of results. Therefore, we can observe that *i)* the traffic between gateway and cluster is increased by 50% (CR from 15 to 10), and *ii)* the one between sensor and gateway is compressed by 10×. Hence, depending on the available bandwidth of the two links, different solutions can be optimal. In our case, deploying the HPCA on the sensors allowed us to reduce the number of gateways in the network from 2 to 1, since the same CAN bus supports up to 500 sensor nodes instead of 50.

VII. Conclusion

This work presents the embedding and acceleration of the History PCA algorithm at both gateway and sensor level for data compression in a structural health monitoring system. The implementations of HPCA on gateways obtain the same performance as the standard PCA, i.e. $CR = 15$ with average $RSNR = 16\,\mathrm{dB}$ and a mean error in spectrum peak tracking lower than the frequency resolution 5 mHz, with 1221× lower memory footprint. Further moving the HPCA to the sensor-node, we achieved the same RSNR and functional performance with a slightly lower compression factor $CR = 10$, which yet allows reducing the traffic on the gateway-sensors links by an order of magnitude.

We also compare different implementation on the two-levels of the SHM network and different gateway platforms, demonstrating that: *i)* the Artik 710 achieves 2.9× faster execution and 1.4× energy reduction, with respect to Raspberry Pi 3. *ii)* A 16 b fixed-point implementation meets the memory

TABLE VI
PERFORMANCE OF ALL THE IMPLEMENTATION ON THE DIFFERENT PLATFORMS. SETTINGS: $B = 1, m = 3$.

				Raspberry Pi 3 (d = 500)				ARTIK 710 (d = 500)				STM32F405RG (d = 100/500)	
				time [ms]		energy [mJ]		time [ms]		energy [mJ]		time [ms]	energy [mJ]
	CR	scalability	max(D)	1-c	4-c	1-c	4-c	1-c	8-c	1-c	8-c	1-c	1-c
GT1	15	38 (Rpi3)	∞	44.6	35.3	18.8	44.0	34.7	13.3	34.4	30.9	n.a.	n.a.
GT2	15	100 (Rpi3)	∞	44.6	13.3	18.8	16.6	34.7	4.9	34.4	11.3	n.a.	n.a
SNS1	15	∞	450	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	40.4/n.a.	11.6/n.a.
SNS2	10	∞	500	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	18.0/569.0	5.2/163.0

constraint of the end-nodes at the cost of a reduction in the CR. *iii)* embedding the HPCA on the sensor permits the full scalability of the application and allows the gateway to manage $10\times$ more sensors maintaining the same data-traffic of the sensor-network without the compression applied. Moreover, reducing the traffic allows to move from a wired sensor-network to a wireless sensor network.

Our future work will focus on moving the computation of the HPCA on a multi-core low power end-node to further reduce the energy required to compress the signals. .

ACKNOWLEDGMENT

The research contribution presented in this paper has been funded by a research grant of ST Microelectronics and by the Emilia-Romagna region Doctoral Program.

REFERENCES

[1] E. sayed ali ahmed and Z. Kamal Aldein Mohammed, "Internet of things applications, challenges and related future technologies," *world scientific news*, 01 2017.

[2] H. Cai, B. Xu, L. Jiang, and A. V. Vasilakos, "Iot-based big data storage systems in cloud computing: Perspectives and challenges," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 75–87, Feb 2017.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct 2016.

[4] X. Liu, J. Cao, W. Song, P. Guo, and Z. He, "Distributed sensing for high-quality structural health monitoring using wsns," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 3, pp. 738–747, March 2015.

[5] Q. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Firework: Data processing and sharing for hybrid cloud-edge analytics," *IEEE Transactions on Parallel & Distributed Systems*, vol. 29, no. 09, pp. 2004–2017, sep 2018.

[6] G. Hackmann, W. Guo, G. Yan, Z. Sun, C. Lu, and S. Dyke, "Cyber-physical codesign of distributed structural health monitoring with wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 63–72, Jan 2014.

[7] G. Tang, K. Wu, and J. Lei, "A distributed and scalable approach to semi-intrusive load monitoring," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 6, pp. 1553–1565, June 2016.

[8] M. Aazam and E. Huh, "Fog computing and smart gateway based communication for cloud of things," in *2014 International Conference on Future Internet of Things and Cloud*, Aug 2014, pp. 464–470.

[9] E. Arslan and T. Kosar, "High-speed transfer optimization based on historical analysis and real-time tuning," *IEEE Transactions on Parallel & Distributed Systems*, vol. 29, no. 06, pp. 1303–1316, jun 2018.

[10] D. Yun, C. Q. Wu, and M. M. Zhu, "Transport-support workflow composition and optimization for big data movement in high-performance networks," *IEEE Transactions on Parallel & Distributed Systems*, vol. 28, no. 12, pp. 3656–3670, dec 2017.

[11] S. Di and F. Cappello, "Optimization of error-bounded lossy compression for hard-to-compress hpc data," *IEEE Transactions on Parallel & Distributed Systems*, vol. 29, no. 01, pp. 129–143, jan 2018.

[12] A. Abdelgawad and K. Yelamarthi, "Structural health monitoring: Internet of things application," in *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Oct 2016, pp. 1–4.

[13] C. A. Tokognon, B. Gao, G. Y. Tian, and Y. Yan, "Structural health monitoring framework based on internet of things: A survey," *IEEE Internet of Things Journal*, vol. 4, no. 3, pp. 619–635, June 2017.

[14] D. Inaudi, "Long-term static structural health monitoring," in *Structures Congress 2010*, 2010, pp. 566–577.

[15] A. Burrello, D. Brunelli, M. Malavisi, and L. Benini, "Enhancing structural health monitoring with vehicle identification and tracking," in *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2020, pp. 1–6.

[16] H.-N. Li, T.-H. Yi, L. Ren, D.-S. Li, and L.-S. Huo, "Reviews on innovations and applications in structural health monitoring for infrastructures," *Struct. Monit. Maint*, vol. 1, no. 1, pp. 1–45, 2014.

[17] C. Ayyildiz, H. E. Erdem, T. Dirikgil, O. Dugenci, T. Kocak, F. Altun, and V. C. Gungor, "Structure health monitoring using wireless sensor networks on structural elements," *Ad Hoc Networks*, vol. 82, pp. 68 – 76, 2019.

[18] M. Reyer, S. Hurlebaus, J. Mander, and O. E. Ozbulut, "Design of a wireless sensor network for structural health monitoring of bridges," in *2011 Fifth International Conference on Sensing Technology*, Nov 2011, pp. 515–520.

[19] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on information theory*, vol. 23, no. 3, pp. 337–343, 1977.

[20] C. R. Farrar and K. Worden, "An introduction to structural health monitoring," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 365, no. 1851, pp. 303–315, 2007.

[21] Z. Su and L. Ye, "An intelligent signal processing and pattern recognition technique for defect identification using an active sensor network," *Smart Materials and Structures*, vol. 13, no. 4, pp. 957–969, jun 2004.

[22] S. Park, J.-J. Lee, C.-B. Yun, and D. J. Inman, "Electro-mechanical impedance-based wireless structural health monitoring using pca-data compression and k-means clustering algorithms," *Journal of Intelligent Material Systems and Structures*, vol. 19, no. 4, pp. 509–520, 2008.

[23] L. Balzano, Y. Chi, and Y. M. Lu, "Streaming pca and subspace tracking: The missing data case," *Proceedings of the IEEE*, vol. 106, no. 8, pp. 1293–1310, 2018.

[24] P. Yang, C.-J. Hsieh, and J.-L. Wang, "History pca: A new algorithm for streaming pca," 2018.

[25] A. Burrello, A. Marchioni, D. Brunelli, and L. Benini, "Embedding principal component analysis for data reduction in structural health monitoring on low-cost iot gateways," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*. ACM, 2019, pp. 235–239.

[26] S. Yin, S. X. Ding, X. Xie, and H. Luo, "A review on basic data-driven approaches for industrial process monitoring," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 11, pp. 6418–6428, 2014.

[27] A. Marchioni, M. Mangia, F. Pareschi, R. Rovatti, and G. Setti, "Subspace energy monitoring for anomaly detection @sensor or @edge," *IEEE Internet of Things Journal*, pp. 1–1, 2020.

[28] S. A. . Family. (2016) Samsung artik 710 iot module. [Online]. Available: https://www.artik.io/modules/artik-710/

[29] R. P. Foundation. (2016) Raspberry pi 3 module b. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/

[30] M. Grossberg, S. Gottipati, I. Gladkova, M. Rabinowitz, P. Alabi, T. George, and A. Pacheco, "A comparative study of lossless compression algorithms on multispectral imager data," in *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery*

XV, vol. 7334. International Society for Optics and Photonics, 2009, p. 733408.

[31] D. Blalock, S. Madden, and J. Guttag, "Sprintz: Time series compression for the internet of things," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 3, p. 93, 2018.

[32] M. Jayawardhana, X. Zhu, R. Liyanapathirana, and U. Gunawardana, "Compressive sensing for efficient health monitoring and effective damage detection of structures," *Mechanical Systems and Signal Processing*, vol. 84, pp. 414–430, 2017.

[33] S. Liu and L. Ch, "Efficient data compression in wireless sensor networks for civil infrastructure health monitoring," in *2006 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, vol. 3. IEEE, 2006, pp. 823–829.

[34] M. A. Alsheikh, S. Lin, D. Niyato, and H.-P. Tan, "Rate-distortion balanced data compression for wireless sensor networks," *IEEE Sensors Journal*, vol. 16, no. 12, pp. 5072–5083, 2016.

[35] M. Wu, L. Tan, and N. Xiong, "Data prediction, compression, and recovery in clustered wireless sensor networks for environmental monitoring applications," *Information Sciences*, vol. 329, pp. 800–818, 2016.

[36] S. Kaisler, F. Armour, J. A. Espinosa, and W. Money, "Big data: Issues and challenges moving forward," in *2013 46th Hawaii International Conference on System Sciences*, Jan 2013, pp. 995–1004.

[37] Z. Zhao, R. Zhao, J. Xia, X. Lei, D. Li, C. Yuen, and L. Fan, "A Novel Framework of Three-Hierarchical Offloading Optimization for MEC in Industrial IoT Networks," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5424–5434, 2020.

[38] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, Aug 2016.

[39] C. M. Gamanayake, L. A. Jayasinghe, B. Ng, and C. Yuen, "Cluster Pruning: An Efficient Filter Pruning Method for Edge AI Vision Applications," *IEEE Journal of Selected Topics in Signal Processing*, pp. 1–1, 2020.

[40] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny Deep Learning on IoT Devices," 2020.

[41] C. M. Sadler and M. Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems*. ACM, 2006, pp. 265–278.

[42] F. Marcelloni and M. Vecchio, "A simple algorithm for data compression in wireless sensor networks," *IEEE communications letters*, vol. 12, no. 6, pp. 411–413, 2008.

[43] M. A. Razzaque, C. Bleakley, and S. Dobson, "Compression in wireless sensor networks: A survey and comparative evaluation," *ACM Trans. Sen. Netw.*, vol. 10, no. 1, pp. 5:1–5:44, Dec. 2013.

[44] M. A. Razzaque and S. Dobson, "Energy-efficient sensing in wireless sensor networks using compressed sensing," *Sensors*, vol. 14, no. 2, pp. 2822–2859, 2014. [Online]. Available: https://www.mdpi.com/1424-8220/14/2/2822

[45] M. Rani, S. B. Dhok, and R. B. Deshmukh, "A systematic review of compressive sensing: Concepts, implementations and applications," *IEEE Access*, vol. 6, pp. 4875–4894, 2018.

[46] H. Mamaghanian, N. Khaled, D. Atienza, and P. Vandergheynst, "Compressed sensing for real-time energy-efficient ecg compression on wireless body sensor nodes," *IEEE Transactions on Biomedical Engineering*, vol. 58, no. 9, pp. 2456–2466, Sep. 2011.

[47] V. Cambareri, M. Mangia, F. Pareschi, R. Rovatti, and G. Setti, "A case study in low-complexity ecg signal encoding: How compressing is compressed sensing?" *IEEE Signal Processing Letters*, vol. 22, no. 10, pp. 1743–1747, Oct 2015.

[48] D. Del Testa and M. Rossi, "Lightweight lossy compression of biometric patterns via denoising autoencoders," *IEEE Signal Processing Letters*, vol. 22, no. 12, pp. 2304–2308, Dec 2015.

[49] C. S. N. Pathirage, J. Li, L. Li, H. Hao, W. Liu, and P. Ni, "Structural damage identification based on autoencoder neural networks and deep learning," *Engineering Structures*, vol. 172, pp. 13 – 28, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141029618302062

[50] I. T. Jolliffe, *Principal Component Analysis*. New York, NY: Springer New York, 1986.

[51] T. Krasulina, "The method of stochastic approximation for the determination of the least eigenvalue of a symmetrical matrix," *USSR Computational Mathematics and Mathematical Physics*, vol. 9, no. 6, pp. 189 – 195, 1969.

[52] E. Oja, "Simplified neuron model as a principal component analyzer," *Journal of Mathematical Biology*, vol. 15, no. 3, p. 267273, 1982.

[53] I. Mitliagkas, C. Caramanis, and P. Jain, "Memory limited, streaming pca," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 2886–2894.

[54] M. Hardt and E. Price, "The noisy power method: A meta algorithm with applications," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2861–2869.

[55] Z. Allen-Zhu and Y. Li, "First efficient convergence for streaming k-pca: A global, gap-free, and near-optimal rate," in *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, 2017, pp. 487–492.

[56] N. Tripuraneni, N. Flammarion, F. Bach, and M. I. Jordan, "Averaging stochastic gradient descent on Riemannian manifolds," in *Proceedings of the 31st Conference On Learning Theory*, ser. Proceedings of Machine Learning Research, S. Bubeck, V. Perchet, and P. Rigollet, Eds., vol. 75. PMLR, 06–09 Jul 2018, pp. 650–687.

[57] M. Brand, "Fast low-rank modifications of the thin singular value decomposition," *Linear Algebra and its Applications*, vol. 415, no. 1, pp. 20 – 30, 2006, special Issue on Large Scale Linear and Nonlinear Eigenvalue Problems.

[58] C. Tang, "Exponentially convergent stochastic k-pca without variance reduction," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 12 393–12 404.

[59] J. R. Bunch and C. P. Nielsen, "Updating the singular value decomposition," *Numerische Mathematik*, vol. 31, p. 111129, 1978.

[60] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Transactions on Computers*, vol. C-23, no. 1, pp. 90–93, Jan 1974.

[61] D. L. Donoho, "For most large underdetermined systems of linear equations the minimal l-norm solution is also the sparsest solution," *Communications on Pure and Applied Mathematics*, vol. 59, no. 6, pp. 797–829, 2006.

[62] M. Mangia, R. Rovatti, and G. Setti, "Rakeness in the design of analog-to-information conversion of sparse and localized signals," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 5, pp. 1001–1014, May 2012.

[63] W. Gander, "Algorithms for the qr decomposition," *Res. Rep*, vol. 80, no. 02, pp. 1251–1268, 1980.

[64] P. Businger and G. H. Golub, "Linear least squares solutions by householder transformatinos, linear algebra, handbook for automatic computation, vol. 2," 1971.

[65] P. Yang, C.-J. Hsieh, and J.-L. Wang, "History pca: A new algorithm for streaming pca," *arXiv*, 2018. [Online]. Available: https://arxiv.org/abs/1802.05447

[66] A. Girolami, D. Brunelli, and L. Benini, "Low-cost and distributed health monitoring system for critical buildings," in *2017 IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS)*, 2017, pp. 1–6.

[67] A. Girolami, F. Zonzini, L. De Marchi, D. Brunelli, and L. Benini, "Modal analysis of structures with low-cost embedded systems," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–4.

[68] G. Bertagnoli, F. Lucà, M. Malavisi, D. Melpignano, and A. Cigada, "A large scale shm system: A case study on pre-stressed bridge and cloud architecture," in *Dynamics of Civil Structures, Volume 2*, S. Pakzad, Ed. Cham: Springer International Publishing, 2020, pp. 75–83.

[69] lis344alh sensor, STMicroelectronics. [Online]. Available: https://www.st.com/en/mems-and-sensors/lis344alh.html

[70] Hts221 sensor, STMicroelectronics. [Online]. Available: https://www.st.com/resource/en/datasheet/hts221.pdf

[71] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online]. Available: http://www.scipy.org/

[72] P. C. Chang, A. Flatau, and S. C. Liu, "Review paper: Health monitoring of civil infrastructure," *Structural Health Monitoring*, vol. 2, no. 3, pp. 257–267, 2003.