Chapter 1

Distributed and Adaptive Source Coding

1.1 Introduction

Advances on the electronics and wireless networking have enabled the appearance of new tiny devices endowed with many different types of sensors and capabilities. They are going to make possible the beginning of new exciting applications that will open doors previously closed to the human being. This nodes are usually small in physical dimensions and operated by battery power. Since in most cases access to the sensors once they have been deployed is extremely hard or in many cases almost impossible, it is easy to understand that any technology that make possible energy savings is welcome. Thus, a big effort has been made by the research and scientific community in order to reduce energy consumption in such networks. In this chapter, a method proposed by Petrovic, Ramchandran and Chou [3] is studied.

An implementation on a real Wireless Sensor Network (WSN) was also carried out. Results are presented in Chapter 4.

The studied algorithm is based on the adaptive signal processing theory as well as on distributed source coding. The main underlying idea consists of taking advantage of the correlation brought about by the spatio-temporal characteristics of the physical medium to reduce the amount of

information requested to the sensors. Furthermore, this scheme is orthogonally different from other studies carried out in the area, allowing a total integration with other energy-aware technologies like packet/data aggregation (reduces the overhead in the network) [4][5][6], efficient information processing [7][8].

Two factors of correlation should be highlighted and, consequently, considered for the correlation tracking algorithm:

- Measures taken by sensors closely located show a similar pattern.
- The signals being sensed (temperature, humidity,...) follow some basic rules of continuity and/or statistical properties.

Dense networks offer data samples highly correlated, because the sensors used are physically placed closely, with the consequent redundance obtained in the measures. An example of this can be a WSN deployed to measure the temperature and humidity in the ecosystem created around a tree, with the sensors measuring the evolution of these parameters in different parts of the tree. Another example can be the recording of audio data [3] as the cry of the whales or even a concert. Audio is a very suitable data source to apply this algorithm, due to the intrinsic presence of redundance (echoes).

Petrovich's algorithm is applied in a network consisting on two types of nodes: many sensing nodes and one or more data-gathering nodes (see Fig. 1.1). The former ones are supposed to be energy constrained, not so the latter. One of the most appealing characteristics of this algorithm is that no data exchange between sensors is needed, the compression takes place in a fully blind manner, with the subsequent saving of energy. These savings are achieved by letting the data gathering node track the correlation structure among nodes and process the information to effect distributed sensor data compression. The correlation structure is determined by using an adaptive prediction algorithm. The sensors, in turn, only need to know the number of bits they are entitled to send to the data-gathering node.

It is obvious the complexity in the decoder is higher than that in the encoder, but it resides on the data-gathering node, which is assumed not to

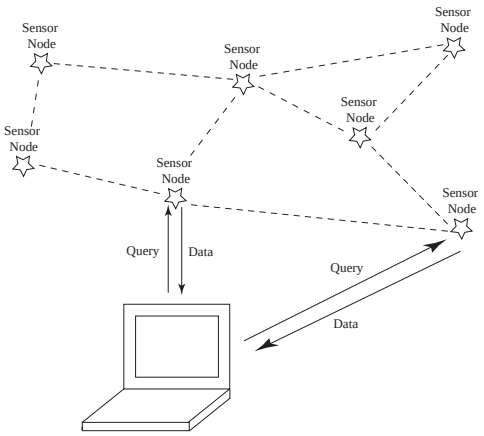

Figure 1.1: *An example of a WSN in which the laptop acts as the sink node gathering the information requested to the sensors.*

be energy constrained.

1.2 Distributed compression

Let us focus now in how distributed compression is achieved. Say we have $n + 1$ nodes, where one of them behaves as a data-gathering node. The algorithm works as follows: when the sink node starts to request information it has no information at all which allows it to perform redundance elimination, thus, it begins asking for uncompressed data. Once it is in possession of data coming from the sensors it is able to reduce the amount of information requested to the sensors, since this information will be partially redundant with the one previously received. The data-gathering node uses this old information to obtain a prediction of the next value it is going to request.

Let us imagine it is turn to query sensor j and we are at time instant k. The desired data can be represented by $X_k^{(j)}$. The corresponding estimate is denoted by $Y_k^{(j)}$ and is calculated by means of the correlation tracking algorithm. This algorithm uses previous measures of sensor j as well as measures belonging to neighboring sensor nodes. All measures are conveniently weighted to yield the optimal estimate in terms of a particular criterion (in this case we chose to minimize the mean square error). Based

on the accuracy of this prediction and some other parameters as the desired probability of error, the sink node will require a specific number of bits (say m) to sensor j which, in turn, will transmit the data sensed (converted by the ADC to n bits) compressed to those m bits. The decoder will have to decode this compressed sequence of bits to the desired data message, $X_k^{(j)}$, for what it uses the value $Y_k^{(j)}$ previously calculated.

The correlation tracking algorithm must address an important issue: data statistics might be time-varying and consequently, the amount of correlation can change. This has two consequences, first, the correlation tracking algorithm can not be static and the coefficients weighting the different data have to be updated with the sufficient frequency. Second, it is mandatory to have one unique underlying codebook that it is not changed among the sensors and can also support multiple compression rates. This concept can be pictured as having several codebooks forming a tree structure as shown in Fig. 1.2.

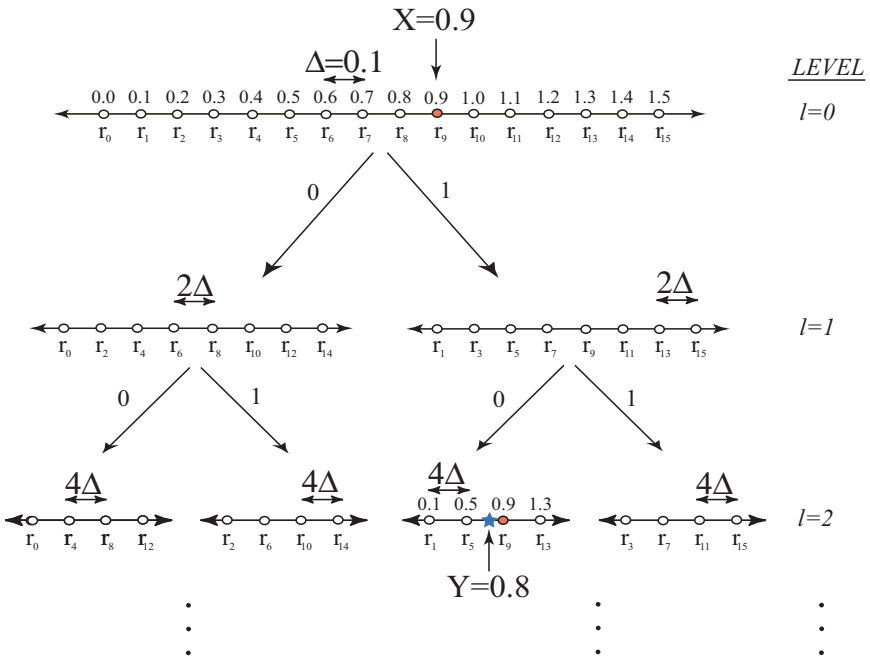

Figure 1.2: *An example for the tree based codebook. The encoder is asked to encode X using 2 bits, so it transmits $\{01\}$ accordingly to the decoder. The decoder will use the bits $\{01\}$ in an ascending order from the least-significative-bit (LSB) to determine the path to the subcodebook to use to decode Y.*

During the remaining of the chapter we will repeatedly talk about codewords and codebooks, so the reader should have a clear understanding of what is meant by these two terms. By codeword we denote the binary representation for each of the symbols coming out from the quantizer. The codebook is just the set of all possible codewords. Using the same nomenclature as in Appendix A, we can say that the codebook is the binary representation of the alphabet whereas the codewords are the binary representations of the letters. Some times, we talk about codebooks at a particular level, this is an abuse of language by which we denote the set of possible codewords in that determinate level. For example, in Figure 1.2, a codeword is represented by r_j ($j \in \{0, \cdots, 15\}$) while the codebook at level 0 is the set $\{r_0, r_1, r_2, r_3, \cdots, r_{14}, r_{15}\}$. In Figure 1.2, the distance between codewords at level i is $2^i\Delta$.

Let us briefly illustrate, now, how encoder and decoder work:

- Encoder: The encoding operation is carried out by the sensing nodes. It is preceded by the computation (in the sink node) of the number of necessary bits i that are going to be requested to the sensors for the data to be compressed. Once this number is computed (based on different parameters as probability of error, accuracy of previous predictions,...) the sensor is asked to send its information compressed down to this number of bits. Suppose the ADC returns the data sensed, $X_k^{(j)}$, with n bits. The mapping from $X_k^{(j)}$ to the compressed message with the specified number of bits can be done through the following deterministic mapping (1.1), resulting in new codewords belonging to a subcodebook at level i.

$$f(X_k^{(j)}) = \text{index}(X_k^{(j)}) mod 2^i \tag{1.1}$$

  What we basically do with this mapping is to keep the i least significative bits of the original data $X_k^{(j)}$. That is, we assume that the estimated value of $X_k^{(j)}$ is sufficiently accurate to predict the overall behavior of the sensed data so that it is only needed a small amount of information from the nodes to precisely determinate the value $X_k^{(j)}$. This way of proceeding is called decoding with side-information, where the side-information is the predicted value of $X_k^{(j)}$, $Y_k^{(j)}$, computed at

the decoder (the interested reader can find more information on this topic in Appendix A).

- Decoder: The decoding operation is, obviously, performed at the data-gathering node. When this node receives the requested data it starts to traverse the tree using the information received to locate the appropriate subcodebook $\mathcal{S}$ among all the possible ones at the level i at which the data has been compressed to. This process starts with the least-significant-bit (LSB) of $f(X_k^{(j)})$. Once the suitable subcodebook has been found, the decoder will use the side-information, $Y_k^{(j)}$, to decode the closest value in $\mathcal{S}$:

$$f(X_k^{(j)}) = \text{argmin}_{r_i \in \mathcal{S}} \| Y_k^{(j)} - r_i \| \tag{1.2}$$

  where r_i stands for the i^{th} codeword in $\mathcal{S}$.

If we study more carefully how the tree structure of subcodebooks is constructed, we can appreciate that at level i, the different codewords within the same subcodebook share the last i bits. This can be directly interpreted as the fact that the estimate $Y_k^{(j)}$ is not distinguishable among words of code belonging to level $i+1$ (or higher) so that it is necessary to have a finer decomposition. Thus, buy asking for the i least significative bits, we are able to rule out some codewords so that our subcodebook is small enough to guarantee a correct decoding from the side-information.

In short, we assure that the prediction $Y_k^{(j)}$ differs in less than $2^{i-1}\Delta$ from $X_k^{(j)}$ and we choose a subcodebook $\mathcal{S}$ whose codewords are at a distance $2^i\Delta$ from each other. In this way, a unique and successful decoding is achieved.

Let us illustrate the coding/decoding process with a simple example. Assume we have the tree structure given by Fig. 1.2, where each codeword in the codebook at level $l = 0$ is composed by four bits (the codebook has sixteen elements). Let us also assume that the data-gathering node has in some way computed the side-information $Y_k^{(j)}$ and the number of bits to which the data has to be compressed is $i = 2$. The process starts when the sink node requests data to sensor j. The sensor node uses its ADC to

obtain a 4-bit value $X_k^{(j)} = 0.9$ corresponding to codeword r_9. The following thing the sensor node does is to use the rule given by (1.1) to find the mapping in the compressed domain: $f(X_k^{(j)}) = 9 \text{mod} 4 = 1$. Thus, the encoder will send the two bits, $\{01\}$, to the k-th sink node. The sink node will make use of the received message to descend by the tree and find the suitable subcodebook $\mathcal{S}$ where to apply (1.2). It starts using the LSB '1' so that it breaks the root codebook (i.e., the codebook at level 0) down to $\{r_1, r_3, r_5, r_7, r_9, r_{11}, r_{13}, r_{15}\}$. Afterwards it uses the second bit in the message '0' to choose codebook $\{r_1, r_5, r_9, r_{13}\}$ as $\mathcal{S}$. It is now when the decoder uses the side-information $Y_k^{(j)}$ conveniently introduced in Eq. (1.2)

$$f(X_k^{(j)}) = \text{argmin}_{r_i \in \mathcal{S}} \{0.7, 0.3, 0.1, 0.5\}$$

to derive r_9 as the decoded codeword, which is exactly the value sensed in the node. Thus, we can see how transmission of the information has been achieved by using only 2 bits instead of 4, as it would have been needed without encoding.

1.3 Correlation tracking

In the previous section we assumed that some information, $Y_k^{(j)}$, correlated to the sensors readings, $X_k^{(j)}$, was available at the decoder for sensor j at time k. The prediction can be done by a linear combination of different measures available at the decoder and can be expressed as Eq. (1.3):

$$Y_k^{(j)} = \sum_{l=1}^{M} \alpha_l X_{k-l}^{(j)} + \sum_{i=1}^{j-1} \beta_i X_k^{(i)} \tag{1.3}$$

We can think of $Y_k^{(j)}$ as a linear prediction based on past values of the sensor whose measure is going to be predicted along with current values of neighboring sensors. Hence, the main objective of the decoder is to derive a good estimate of $X_k^{(j)}$ for each sensor j.

To find the values α_l and β_i which minimize the mean square error (MSE), a mathematical problem must be addressed. Let us start by representing the prediction error as a random variable, $N_j = Y_k^{(j)} - X_k^{(j)}$. We can expand

the mean square error as:

$$\begin{aligned} E\left[N_j^2\right] &= E\left[\left(X_k^{(j)} - \left(\sum_{l=1}^{M}\alpha_l X_{k-l}^{(j)} + \sum_{i=1}^{j-1}\beta_i X_k^{(i)}\right)\right)^2\right] \\ &= E\left[X_k^{(j)^2}\right] - 2\sum_{l=1}^{M}\alpha_l E\left[X_k^{(j)}X_{k-l}^{(j)}\right] \\ &\quad -2\sum_{i=1}^{N}\beta_i E\left[X_k^{(j)}X_k^{(i)}\right] + 2\sum_{l=1}^{M}\sum_{i=1}^{j-1}\alpha_l\beta_i E\left[X_{k-l}^{(j)}X_k^{(i)}\right] \\ &\quad + \sum_{l,h=1}^{M}\alpha_l\alpha_h E\left[X_{k-l}^{(j)}X_{k-h}^{(j)}\right] + \sum_{i,h=1}^{j-1}\beta_i\beta_h E\left[X_k^{(i)}X_k^{(h)}\right] \quad (1.4) \end{aligned}$$

Now, if we assume that $X_k^{(j)}$ and $X_k^{(i)}$ are pairwise jointly wide sense stationary for $i = 1, \ldots, j-1$ then we can rewrite the mean square error as:

$$E[N_j^2] = r_{x^j x^j}(0) - 2\mathbf{P}_j^T\mathbf{\Gamma}_j + \mathbf{\Gamma}_j^T R_{zz}^j \mathbf{\Gamma}_j \quad (1.5)$$

where the superscript T stands for the transpose and

$$\mathbf{\Gamma}_j = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_M & \beta_1 & \beta_2 & \cdots & \beta_{j-1} \end{bmatrix}^T,$$

$$\mathbf{P}_j = \begin{bmatrix} r_{x^j x^j}(1) & r_{x^j x^j}(2) & \cdots & r_{x^j x^j}(M) & r_{x^j x^1}(0) & r_{x^j x^2}(0) & \cdots & r_{x^j x^{j-1}}(0) \end{bmatrix}^T$$

and we use the notation $r_{x^j x^i}(l) = E[X_k^j X_{k+l}^i]$. We can express R_{zz}^j as:

$$R_{zz}^j = \begin{bmatrix} R_{x^j x^j} & R_{x^j x^i} \\ R_{x^j x^i}^T & R_{x^i x^i} \end{bmatrix} \quad (1.6)$$

where, in turn, $R_{x^j x^j}$ is given as:

$$R_{x^j x^j} = \begin{bmatrix} r_{x^j x^j}(0) & r_{x^j x^j}(1) & \cdots & r_{x^j x^j}(M-1) \\ r_{x^j x^j}(1) & r_{x^j x^j}(0) & \cdots & r_{x^j x^j}(M-2) \\ \vdots & \vdots & \ddots & \vdots \\ r_{x^j x^j}(M-1) & r_{x^j x^j}(M-2) & \cdots & r_{x^j x^j}(0) \end{bmatrix} \quad (1.7)$$

and $R_{x^j x^i}$ and $R_{x^i x^i}$ are given as:

$$R_{x^j x^j} = \begin{bmatrix} r_{x^j x^1}(1) & r_{x^j x^2}(1) & \cdots & r_{x^j x^{j-1}}(1) \\ r_{x^j x^1}(2) & r_{x^j x^2}(2) & \cdots & r_{x^j x^{j-1}}(2) \\ \vdots & \vdots & \ddots & \vdots \\ r_{x^j x^1}(M) & r_{x^j x^2}(M) & \cdots & r_{x^j x^{j-1}}(M) \end{bmatrix} \tag{1.8}$$

$$R_{x^i x^i} = \begin{bmatrix} r_{x^1 x^1}(0) & r_{x^1 x^2}(0) & \cdots & r_{x^1 x^{j-1}}(0) \\ r_{x^2 x^1}(0) & r_{x^2 x^2}(0) & \cdots & r_{x^2 x^{j-1}}(0) \\ \vdots & \vdots & \ddots & \vdots \\ r_{x^{j-1} x^1}(0) & r_{x^{j-1} x^2}(0) & \cdots & r_{x^{j-1} x^{j-1}}(0) \end{bmatrix} \tag{1.9}$$

To find the optimal set of coefficients (represented by $\mathbf{\Gamma}_j$) that minimizes the mean square error, we differentiate Eq. (1.5) with respect to $\mathbf{\Gamma}_j$ to obtain:

$$\begin{aligned} \frac{\partial E[N_j^2]}{\partial \mathbf{\Gamma}_j} &= \frac{\partial \left(r_{x^j x^j}(0) - 2\mathbf{P}_j^T \mathbf{\Gamma}_j + \mathbf{\Gamma}_j^T R_{zz}^j \mathbf{\Gamma}_j \right)}{\partial \mathbf{\Gamma}_j} \\ &= \frac{\partial \left(-2\mathbf{P}_j^T \mathbf{\Gamma}_j \right)}{\partial \mathbf{\Gamma}_j} + \frac{\partial \left(\mathbf{\Gamma}_j^T R_{zz}^j \mathbf{\Gamma}_j \right)}{\partial \mathbf{\Gamma}_j} \\ &= -2\mathbf{P}_j + \left[R_{zz}^{jT} + R_{zz}^j \right] \mathbf{\Gamma}_j \\ &= -2\mathbf{P}_j + 2R_{zz}^j \mathbf{\Gamma}_j \end{aligned} \tag{1.10}$$

Where basic matrix calculus has been employed:

$$\frac{\partial (Ax+b)^T C\,(Dx+e)}{\partial x} = A^T C\,(Dx+e) + D^T C^T\,(Ax+b) \tag{1.11}$$

Please note that in the formula above (1.11) upper case letters denote matrixes while vectors are written in the lower case.

Setting (1.10) equals zero and solving, we obtain the standard Wiener estimate:

$$\mathbf{\Gamma}_{j,opt} = R_{zz}^{-1,j} \mathbf{P}_j \tag{1.12}$$

If the assumption of stationarity held, then the data gathering could request for uncoded data from all the sensors for the first K rounds of requests and construct the correlation matrixes for their subsequent use in (1.12). By doing this we would already have tracked the behavior of the system and we

could employ the obtained set of coefficients for computing the side information for each future round of requests. Thus, it would be only necessary to calculate the standard Wiener estimate once, what would be really convenient given the extreme computational complexity of this calculus.

In practice, however, the statistics of the data may be (and actually they are) time varying and as a result, the coefficient vector, $\mathbf{\Gamma}_j$, must be continuously adjusted to minimize the mean square error. One method of doing this is to move $\mathbf{\Gamma}_j$ in the opposite direction of the gradient of the objective function (i.e., the mean squared error) for each new sample received during round $k+1$ (this method known in the literature as the 'Steepest Descent Method'):

$$\mathbf{\Gamma}_j^{(k+1)} = \mathbf{\Gamma}_j^{(k)} - \mu \nabla_j^{(k)} \tag{1.13}$$

where $\nabla_j^{(k)}$ is given by Eq. (1.10) and μ represents the step size of the Steepest Descent Method. The goal of this approach is to descend to the global minima of the objective function. We are assured that such a minima exists because the objective function is convex. In fact, it has been shown that if μ is chosen correctly then (1.13) will converge to the optimal solution.

From (1.10) and (1.13) can be shown that the coefficient vector should be updated following the rule:

$$\mathbf{\Gamma_j^{(k+1)}} = \mathbf{\Gamma}_j^{(k)} - \frac{1}{2}\mu \left(-2\mathbf{P}_j + 2R_{zz}^j \mathbf{\Gamma}_j^{(k)}\right) \tag{1.14}$$

However, in practice, the data gathering node will not have knowledge of $\mathbf{P}_j$ and R_{zz}^j due to the computational complexity required for obtaining them. Hence, it will be necessary to provide the algorithm with an efficient method for estimating $\mathbf{P}_j$ and R_{zz}^j. One standard estimate is to use $\mathbf{P}_j =$

$X_k^{(j)}\mathbf{Z}_{k,j}$ and $R_{zz} = \mathbf{Z}_{k,j}\mathbf{Z}_{k,j}$ with

$$\mathbf{Z}_{k,j} = \begin{bmatrix} X_{k-1}^{(j)} \\ X_{k-2}^{(j)} \\ \cdots \\ X_{k-M}^{(j)} \\ X_k^{(1)} \\ X_k^{(2)} \\ \cdots \\ X_k^{(j-1)} \end{bmatrix}$$

introducing these new terms, Eq. (1.14) remains as

$$\begin{aligned} \mathbf{\Gamma}_j^{(k+1)} &= \mathbf{\Gamma}_j^{(k)} - \mu\mathbf{Z}_{k,j}(-X_k^{(j)} + \mathbf{Z}_{k,j}^T\mathbf{\Gamma}_j^{(k)}) \\ &= \mathbf{\Gamma}_j^{(k)} + \mu\mathbf{Z}_{k,j}N_{k,j} \end{aligned} \tag{1.15}$$

where the second equality follows from the fact that $Y_k^{(j)} = \mathbf{Z}_{k,j}^T\mathbf{\Gamma}_j^{(k)}$ and $N_{k,j} = X_k^{(j)} - Y_k^{(j)}$.

In practice the formulas above yield the following practical equations (well known in the adaptive filtering literature as the Least-Mean-Squares (LMS) algorithm):

1. $Y_k^{(j)} = \mathbf{\Gamma}_j^{(k)T}\mathbf{Z}_{k,j}$
2. $N_{k,j} = X_k^{(j)} - Y_k^{(j)}$
3. $\mathbf{\Gamma}_j^{(k+1)} = \mathbf{\Gamma}_j^{(k)} + \mu\mathbf{Z}_{k,j}N_{k,j}$

To use this algorithm, the data-gathering node will start asking the sensing nodes for sending their data uncompressed during the first K rounds. By doing this we ensure that the algorithm converges. Once this phase has been carried out, we can consider that we have tracked the correlation structure existing between the different nodes, so we can start to ask for compressed data. This is done in two steps (as already shown): first, computing the prediction of the data to be requested, for what we know that $Y_k^{(j)} = \mathbf{\Gamma}_j^{(k)T}\mathbf{Z}_{k,j}$ (recall that the statistics of the sources can change with the time, so the vector of coefficients has to be continuously updated). The

second step mentioned before, consists of obtaining the number of bits for the data to be compressed to.

The decoder, in turn, will decode $Y_k^{(j)}$ to the closest codeword in the sub-codebook $\mathcal{S}$ as has already been seen, yielding $\hat{X}_k^{(j)}$. From section 1.2 we know that while $2^{i-1}\Delta > |N_{k,j}|$ and the encoder encodes $X_k^{(j)}$ with i bits, then, $\hat{X}_k^{(j)} = X_k^{(j)}$ and no decoding errors are made. However, if $|N_{k,j}| > 2^{i-1}\Delta$ then a decoding error will occur. This fact can be used to try to find a bound for the error committed during the prediction process. The most straightforward method is to apply Chebyshev's inequality

$$P\left[|N_{k,j}| > 2^{i-1}\Delta\right] \leq \frac{\sigma_{N_j}^2}{\left(2^{i-1}\Delta\right)^2} \tag{1.16}$$

We adopt the realistic assumption that $N_{k,j}$ is zero-mean and variance $\sigma_{N_j}^2$. Thus, reading from Eq. (1.16), we deduce that we can take for the probability of error P_e any value greater or equal than $\frac{\sigma_{N_j}^2}{\left(2^{i-1}\Delta\right)^2}$ so that we can be sure that $P\left[|N_{k,j}| > 2^{i-1}\Delta\right]$ is fulfilled. We obviously take the equal in the inequality so that the expression of the probability of error remains as

$$P_e = \frac{\sigma_{N_j}^2}{\left(2^{i-1}\Delta\right)^2} \tag{1.17}$$

from where we can work the value of i out

$$i = \frac{1}{2}\log_2\left(\frac{\sigma_{N_j}^2}{\Delta^2 P_e}\right) + 1 \tag{1.18}$$

as the number of bits necessaries to ensure a probability of error less or equal than P_e. Note that it is not necessary to be over-conservative when choosing P_e because Chebyshev's inequality is a loose bound.

If we look thoroughly at expression (1.18) we can appreciate the presence of a new term not taken into account yet: the variance $\sigma_{N_j}^2$. This means that the data gathering node must also maintain an estimate of $\sigma_{N_j}^2$. After the initialization module (K iterations long), the data-gathering node can initialize $\sigma_{N_j}^2$ as

$$\sigma_{N_j}^2 = \frac{1}{K-1}\sum_{i=1}^{K} N_{k,j}^2 \tag{1.19}$$

However, in the main module, the variance can be computed and updated in a weighted way (filtered estimate) as

$$\sigma^2_{N_j,new} = (1-\gamma)\sigma^2_{N_j,old} + \gamma N^2_{k,j} \tag{1.20}$$

The choice of a filtered estimate is consequent with the time varying properties stated before, so we can adapt to changes in the statistics. γ is known as the "forgetting factor" and is a key value for the capacity of adapting to changes in the statistics.

Beyond our scope stands the correction/detection of errors, even though, two possible policies are suggested:

a) To detect errors, the use of a cyclic redundancy check (CRC) is proposed. In this way, each sensor would be entitled to send a CRC formed with its last m readings. Afterwards, the data-gathering node will perform its own CRC, in case both of them do not match, the data-gathering node will decide between dropping the m last readings or asking for their retransmission.

b) Another option is to use error-correction codes, as could be the case of using Reed-Solomon codes [9].

1.4 Querying and reporting algorithm

In this section, the algorithm to be implemented in the encoder and decoder is reported. Note that, at the beginning of the algorithm, the data-gathering node must gather enough information to track the correlation structure, what is achieved by performing K rounds of readings (note that K should be chosen large enough to allow the LMS algorithm convergence).

The data-gathering node should alternate the requests for "uncompressed" data among the nodes to ensure that every single node wastes the same amount of energy.

1.4.1 Data gathering node algorithm

The provided pseudocode basically expresses in programming language what has been analyzed along the previous sections. The only novelty introduced here is that, when carrying out the main module, it is necessary to ask a sensor for its full uncompressed data to keep track of the real data ensuring that the prediction is not producing erroneous estimates. However, the request for uncoded data is alternated between the different sensors, so that we can assure that the waste of energy is shared between sensors.

Once taken into account the previous considerations the pseudocode for the data-gathering node is reported:

Pseudocode for data gathering node:

Initialization:

for $(i = 0; i < K; i++)$
  for $(j = 0; j < num\_sensors; j++)$
    Ask sensor j for its uncoded reading
  end
  for each pair of values i, j
    update correlation parameters, using LMS and (1.19) equations.
  end
end

Main Loop:

for $(k = K; k < N; k++)$
  Request a sensor for uncoded reading
  for each remaining sensor
    determine number of bits, i, to request for using Eq. (1.18)
    request for i bits
  end
  Decode data for each sensor.
  Update correlation parameters for each sensor.
end

1.4.2 Sensing nodes algorithm

Pseudocode for the sensor nodes is given below.

Pseudocode for sensor nodes:

for each request
  Extract i from the request
  Get $X[n]$ from ADC
  Transmit $n \text{mod} 2^i$
end