UNIVERSITE LIBRE DE BRUXELLES
Faculte Des Sciences
Departement D'Informatique
Machine Learning Group

Learning in Wireless Sensor Networks for Energy-Efficient Environmental Monitoring

Yann-Aël Le Borgne

PhD Thesis

This thesis has been written under the supervision of Prof. Gianluca Bontempi.

The members of the jury are:

- Prof. Gianluca Bontempi (Université Libre de Bruxelles, Belgium)
- Doctor Jean-Michel Dricot (Université Libre de Bruxelles, Belgium)
- Prof. Guy Latouche (Université Libre de Bruxelles, Belgium)
- Prof. Tom Lenaerts (Université Libre de Bruxelles, Belgium)
- Prof. Marco Saerens (Université Catholique de Louvain, Belgium)
- Prof. Duc A. Tran (University of Massachusetts, Boston, USA)
- Prof. Alfredo Vaccaro (University of Sannio, Italy)

Acknowledgments

A PhD is a thrilling experience, allowing to go deep in the details of many research issues. It is also a great opportunity to meet passionate people, and to discuss for hours about the best solution to a given problem. The nicest thing is that other people often do not see the problems the way as you do. Hence, the number of hours spent at arguing can be quite long. This in most cases actually leads to interesting ideas.

This PhD gave me the chance to meet many such people, without whom the work presented in this thesis would probably be much different.

First of all, I am very much thankful to Professor Gianluca Bontempi. As a supervisor, he brought many interesting inputs into the work presented in this thesis. He also involved me into research project writing, which gave me the chance to meet people with other backgrounds, such as ethologists, geographers, or health security inspectors. His endless energy to connect people and ideas has been highly stimulating.

Most of the people who otherwise inspired the contributions presented in this thesis were met in a summer school on the theme of wireless sensor networks and smart objects, which took place at the beginning of my PhD. I had the chance to meet in this place some of the most motivated and inspiring individuals in this field. In particular, it was the occasion to meet Silvia Santini, with whom the design of the Adaptive Model Selection algorithm, presented as the first contribution of this thesis, was possible. Friedmann Mattern, Kay Römer, Koen Langendoen, Joe Polastre, Robert Szewcyk, Muneeb Ali, David Merril, and many others also provided inspiring ideas in this thesis.

I am also thankful to the master's students I supervised during my PhD. Mehdi Moussaid, Mathieu Van Der Haegen, and Sylvain Raybaud all discussed innovative ideas to the research topic of learning and wireless sensor networks. I greatly acknowledge these students for their motivation, and their efforts to bring their work to be published.

My colleagues in the Machine Learning Group of the ULB have been great research fellows, with whom many discussions on different research topics were possible. Benjamin Haibe-Kains with microarray data classification issues, Abhilash Miranda with multivariate analysis and the PCA, Patrick Meyer with the bias/variance tradeoff in feature selection, Jean-Michel Dricot with wireless networking and entropy, Olivier Caelen with model selection, and Catharina Olsen on causality issues. I also deeply thank them for proofreading the successive drafts for this thesis.

During this stay in Brussels, I also had the chance to meet many of the people from the Iridia Artificial Intelligence laboratory of the ULB. The Iridia Lab is incredibly lively, and most of the people I met there truly inspired my work. I wish to thank, in alphabetical order, Prasannah Balaprakash, Hugues Bersini, Mauro Birattari, Alexandre Campo, Anders Christensen, Marco Dorigo, Max Manfrin, Marco Saerens and Francisco Santos.

Finally, beyond this research life, I found in Brussels a very interesting city where people have especially different backgrounds. Brussels is the heart of the European Union, and therefore gathers people from the whole European Community. Belgium is nonetheless deeply rooted in its traditions, which makes the cultural life of Brussels quite rich. It is difficult to mention all the people who made me very keen on Brussels. My first thanks go to Alexandre, Barth, Nathalie, Hervé, Eric, Mehdi, Nanou and Valérie. I am however thankful to many other people, and especially to my family, which made possible this work in the first place.

Financial Support

The work presented in this thesis was supported by the COMP^{2}2SYS project, sponsored by the Human Resources and Mobility program of the European Community (MEST-CT-2004-505079), by the PIMAN project, supported by the Institute for the Encouragement of Scientific Research and Innovation of Brussels (IRSIB), and by the OASIS project, sponsored by the Belgian Science Policy.

Abstract

Wireless sensor networks form an emerging class of computing devices capable of observing the world with an unprecedented resolution, and promise to provide a revolutionary instrument for environmental monitoring. Such a network is composed of a collection of battery-operated wireless sensors, or sensor nodes, each of which is equipped with sensing, processing and wireless communication capabilities. Thanks to advances in microelectronics and wireless technologies, wireless sensors are small in size, and can be deployed at low cost over different kinds of environments in order to monitor both over space and time the variations of physical quantities such as temperature, humidity, light, or sound.

In environmental monitoring studies, many applications are expected to run unattended for months or years. Sensor nodes are however constrained by limited resources, particularly in terms of energy. Since communication is one order of magnitude more energy-consuming than processing, the design of data collection schemes that limit the amount of transmitted data is therefore recognized as a central issue for wireless sensor networks.

An efficient way to address this challenge is to approximate, by means of mathematical models, the evolution of the measurements taken by sensors over space and/or time. Indeed, whenever a mathematical model may be used in place of the true measurements, significant gains in communications may be obtained by only transmitting the parameters of the model instead of the set of real measurements. Since in most cases there is little or no a priori information about the variations taken by sensor measurements, the models must be identified in an automated manner. This calls for the use of machine learning techniques, which allow to model the variations of future measurements on the basis of past measurements.

This thesis brings two main contributions to the use of learning techniques in a sensor network. First, we propose an approach which combines time series prediction and model selection for reducing the amount of communication. The rationale of this approach, called *adaptive model selection*, is to let the sensors determine in an automated manner a prediction model that does not only fits their measurements, but that also reduces the amount of transmitted data.

The second main contribution is the design of a *distributed approach for modeling sensed data*, based on the principal component analysis. We first show that the sensor measurements can be transformed along a routing tree in such a way that (i) most of the variability in the measurements is retained, and (ii) the network load sustained by sensor nodes is reduced and more evenly distributed. We then show that approximations to the principal components can be computed in a distributed way. These methods allow to truly distribute the principal component analysis, and find applications not only for approximated data collection tasks, but also for event detection or recognition tasks.

Résumé

Les réseaux de capteurs sans fil forment une nouvelle famille de systèmes informatiques permettant d'observer le monde avec une résolution sans précédent. En particulier, ces systèmes promettent de révolutionner le domaine de l'étude environnementale. Un tel réseau est composé d'un ensemble de capteurs sans fil, ou unités sensorielles, capables de collecter, traiter, et transmettre de l'information. Grâce aux avancées dans les domaines de la microélectronique et des technologies sans fil, ces systèmes sont à la fois peu volumineux et peu coûteux. Ceci permet leurs deploiements dans différents types d'environnements, afin d'observer l'évolution dans le temps et l'espace de quantités physiques telles que la température, l'humidité, la lumière ou le son.

Dans le domaine de l'étude environnementale, les systèmes de prise de mesures doivent souvent fonctionner de manière autonome pendant plusieurs mois ou plusieurs années. Les capteurs sans fil ont cependant des ressources limitées, particulièrement en terme d'énergie. Les communications radios étant d'un ordre de grandeur plus coûteuses en énergie que l'utilisation du processeur, la conception de méthodes de collecte de données limitant la transmission de données est devenue l'un des principaux défis soulevés par cette technologie.

Ce défi peut être abordé de manière efficace par l'utilisation de modèles mathématiques modélisant l'évolution spatiotemporelle des mesures prises par les capteurs. En effet, si un tel modèle peut être utilisé à la place des mesures, d'importants gains en communications peuvent être obtenus en utilisant les paramètres du modèle comme substitut des mesures. Cependant, dans la majorité des cas, peu ou aucune information sur la nature des mesures prises par les capteurs ne sont disponibles, et donc aucun modèle ne peut être *a priori* défini. Dans ces cas, les techniques issues du domaine de l'apprentissage machine sont particulièrement appropriées. Ces techniques ont pour but de créer ces modèles de façon autonome, en anticipant les mesures à venir sur la base des mesures passées.

Dans cette thèse, deux contributions sont principalement apportées permettant l'application de techniques d'apprentissage machine dans le domaine des réseaux de capteurs sans fil. Premièrement, nous proposons une approche qui combine la prédiction de série temporelle avec la sélection de modèles afin de réduire la communication. La logique de cette approche, appelée *sélection de modèle adaptive*, est de permettre aux unités sensorielles de determiner de manière autonome un modèle de prédiction qui anticipe correctement leurs mesures, tout en réduisant l'utilisation de leur radio.

Deuxièmement, nous avons conçu une *approche permettant de modéliser de façon distribuée les mesures collectées*, qui se base sur l'analyse en composantes principales (ACP). Nous montrons d'abord que les mesures des capteurs peuvent être transformées le long d'un arbre de routage, de façon à ce que (i) la majeure partie des variations dans les mesures des capteurs soient conservées, et (ii) la charge réseau soit réduite et mieux distribuée. Nous montrons ensuite qu'une approximation des composantes principales peut être calculée de manière distribuée. Ces méthodes permettent de véritablement distribuer l'ACP, et peuvent être utilisées pour des applications impliquant la collecte de données, mais également pour la détection ou la classification d'événements.

Contents

Chapter 1

Introduction

In wireless sensor networks, communication is among the most energy-consuming task for a wireless sensor. This thesis focuses on the design of learning techniques that trade data accuracy with communication by means of prediction models. Since sensor network measurements are very often correlated, we show that prediction models can often significantly reduce the communication while causing little loss in the accuracy of the measurements.

Wireless sensor networks (WSN) form an emerging class of networks able to monitor environments with high spatiotemporal accuracy. The network is composed of tiny devices known as *wireless sensors* or *motes*, endowed with a microprocessor, a memory, a radio, a battery, and one or more sensors such as temperature, humidity, light or sound sensors [3]. Figure 1.1(a) gives an illustration of a typical wireless sensor platform used in research (Tmote), and of an integrated silicon design (Deputy Dust) [90]. The transmission of data from a WSN to an observer raises numerous issues: wireless sensors are constrained by limited resources, in terms of energy, network data throughput, and computational power. The communication module is a particularly constrained resource since the amount of data that can be routed out of the network is inherently limited by the network capacity. Also, wireless communication is an energy consuming task, identified in many situations as the primary factor of lifetime reduction [3].

This thesis investigates how machine learning algorithms can be used to reduce the amount of data transmitted over the network. The main motivation is that sensor network data are very often correlated both over space and time. Machine learning algorithms can be used to detect these redundancies, and to represent them by means of mathematical models. The use of mathematical models instead of the raw data can allow to substantially reduce the amount of data transmitted in the network, and thus to extend application lifetime.

A typical scenario for a sensor network consists in placing a set of wireless sensors in an environment, such as a field, a forest or a town, and to use the collected measurements to monitor, detect, or track the evolution of a phenomenon over space and time. The network is usually connected to a base station by means of a routing tree, such as illustrated in Figure 1.1(b). The base station allows to centralize the data collected from the network, and acts as a gateway between the sensor network and observers. It may be connected to the Internet, which allows the remote observation of the phenomenon monitored by the sensor network.

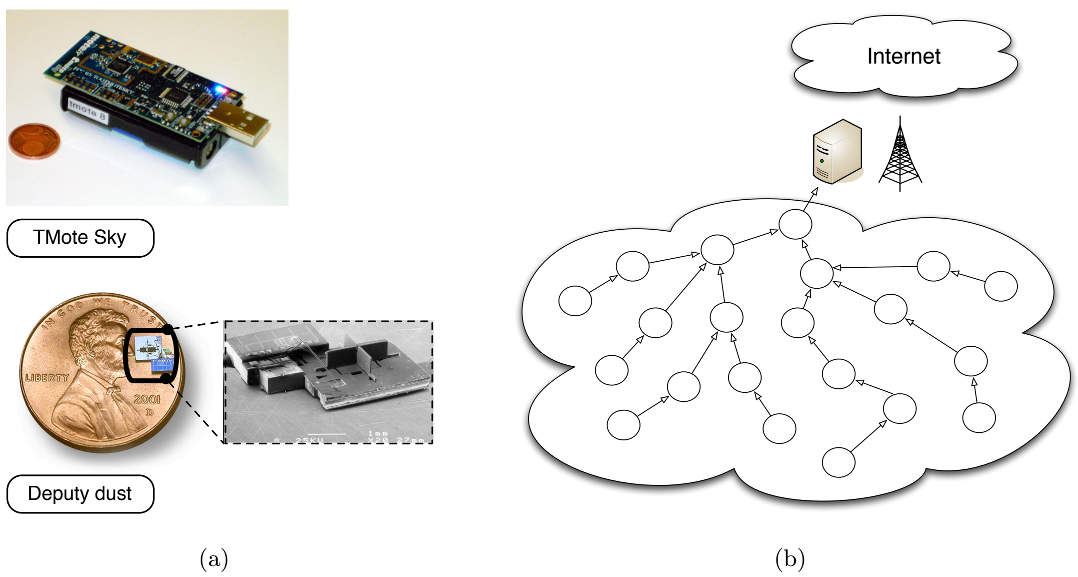


Figure 1.1: (a) Size of wireless sensor nodes in comparison to coin sizes. (b) Multi-hop network architecture for an environmental monitoring application. A routing tree connects the sensor nodes to a base station, which connects the network to the Internet.

1.1 Environmental monitoring

A promising application for WSNs is environmental monitoring, where their use is expected to revolutionize the quality of scientific inquiries for field biologists or ecologists for example [75, 24]. The reasonably low cost and low infrastructure requirements of a WSN enable dense deployments over large areas, providing data at spatiotemporal scales that were previously impossible to obtain. The wireless capabilities of the sensor nodes also allow the remote monitoring, possibly in real-time, of the evolution of a phenomenon. The WSN may be deployed in hostile or inaccessible regions, such as rain forests or volcanoes, and real-time monitoring allows end-users to rapidly react to events. Finally, the small size of the sensor nodes considerably reduces the disturbances caused by the monitoring systems on the environment studied, allowing the monitoring of secretive animals for example.

Thanks to these advantages, sensor networks can be used for collecting data for a wide variety of applications. Two representative projects in environmental monitoring are the Redwood trees and Leach Storm Petrels projects [86, 59]. Redwood trees are among the highest on Earth, and reach heights of around one hundred meters. In 2003, a network of 70 sensor nodes was deployed along one of them in the middle of the Redwood forest in California [86]. By collecting temperature, humidity and light measurements every minute for a forty-four day period, the network provided biologists with data that precisely characterized the microclimate variations between the root and the tree canopy.

The Leach Storm Petrels project was a habitat monitoring project jointly carried out by the University of California, Berkeley, Intel and the College of the Atlantic [59]. Between 2002 and 2003, three networks with sizes ranging from 32 nodes to 147 nodes were deployed inside and outside the burrows of Leach's Storm Petrel, an endangered bird species that forms large colonies on the island during the breeding season. Data were retrieved every five minutes, and remotely made available on an Internet website thanks to the ad-hoc network formed on the island, which connected to the Internet via a satellite communication link. The deployment allowed to retrieve several hundred of thousands of measurements, enabling ethologists to precisely observe the conditions required by these animals during their breeding period.

Other examples of prototypical deployments for environmental monitoring include for instance volcano monitoring [92], glacier displacement [66], or precision agriculture [60] to name a few. In all these applications, the goal of a sensor network is to provide information about the characteristics of a phenomenon in an environment. The interest of the observer ranges from the collection of all measurements from the network at regular time intervals, to the detection, recognition, or tracking of animals in real-time for example.

For many envisioned applications, sensor network deployments make sense only if they can run unattended for many months or even years. The use of renewable energy resources is however rarely viable, and battery replacement is a costly maintenance operation. The energy resources of wireless nodes are therefore in most cases limited to the finite amount of energy stored in their batteries at the moment of the network deployment. This makes the design of energy-efficient data collection strategies a primary concern for extending the lifetime of the wireless sensor network.

1.2 Machine learning for energy-efficient monitoring

Among the different tasks that must be performed by a sensor node, the radio use is by far the most expensive in terms of energy consumption. On a typical sensor node such as the Telos mote, the radio expends one order of magnitude more energy than the CPU over an equivalent length of time [73, 15]. The lifetime of a sensor node therefore highly depends on how frequently the radio is used. If continuously powered, the lifetime of a Telos is typically five days. A central issue in sensor networks thus consists in reducing the amount of communication in the network, while providing the observer with the requested information.

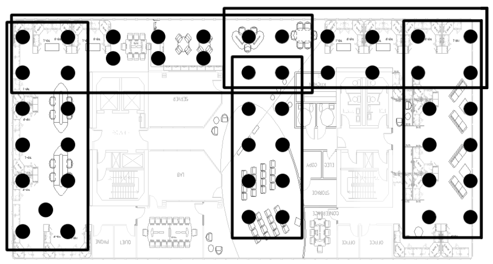
The position advocated in this thesis is that machine learning techniques can efficiently address this challenge. Machine learning techniques in computer science consist in inferring a *prediction model* , on the basis of a set of observations. The model is in most cases a parametric function, which allows to predict the value of a variable, called output, given a set of other variables, called inputs.

In the domain of sensor networks, the use of learning techniques is attractive for a number of reasons. First, sensor data are very often correlated both over space and time. For example, outdoor temperatures typically follow consistent diurnal and seasonal patterns, and at any moment in time, their variations are unlikely to vary greatly within a local region. Learning techniques can be used to detect and model the relationships existing between sensor data, both at the temporal and spatial scales. The use of the models in place of the raw measurements can greatly reduce the amount of data transmitted in the network.

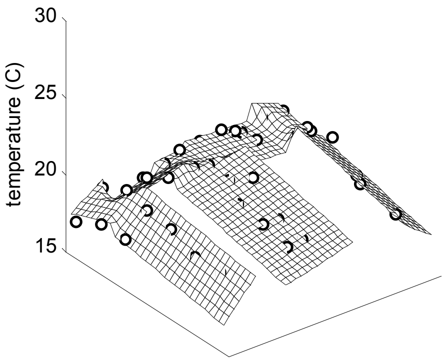
Second, approximations of the sensor measurements can usually be tolerated by the observer. For example, in climatic studies on plant growth, it is often sufficient to collect temperature and humidity measurements within $\pm 0.5°$ and $\pm 2\%$ of the true quantities [19]. One feature of learning techniques is that they can adapt to different levels of accuracy, by using models with varying degree of complexity. The complexity of a model typically depends on its number of parameters. When approximations can be tolerated, simple models with fewer parameters may be used to further reduce the amount of communication.

Finally, the interest of the observer is not necessarily in the measurements, but in *higher level* information such as the types or number of animals present in the monitored environment for example. The extraction of such high-level information typically requires to fuse and transform the measurements from different sensors. Learning techniques provide an effective way to determine how the measurements can be combined in order to infer the requested information. In some cases, it is possible to combine the measurements within the network, thus avoiding the transmission of all the measurements to the base station. These strategies, referred to as *in-network data aggregation*, are among the most promising ones for efficiently extracting information from a sensor network [25, 56].

We briefly illustrate the potential of predictive models in Figure 1.2, using temperature data collected in a WSN experiment carried by the Intel laboratory at Berkeley in 2003 [39]. The network is composed of 47 sensors, deployed throughout a large office. The temperature variations can vary greatly across the office. However, they locally present linear trends, which can be effectively captured by linear models. Each model is here a function of spatial coordinates that are linearly mapped to temperature measurements. Five regions are first delimited, in which the temperature variations are observed to be nearly linear over space, cf. Figure 1.2(a) . The resulting modeling is illustrated in 1.2(b). Each model requires only three parameters to represent the variations within a region, and provides the observer not only with approximations of the measurements at the sensor's locations, but also at any location within a given region.



(a)



(b)

Figure 1.2: Illustration of the ability of models to approximate the variations of temperature measurement throughout a large office (Guestrin et al. [30]). Using the model parameters in place of the raw measurements significantly reduces the amount of communication. (a) Map of the placement of 47 sensors in the office. Five regions are delimited, in which a different model is used to approximate the collected measurements. (b) Example of modeling obtained in each of the five regions, using linear models.

Modeling of sensor data by means of spatial linear models was investigated in [30], and will be presented in detail in Chapter 3. We briefly use it here as an introductory example since, besides illustrating the potential of models for compactly representing the variations of measurements in an environment, it also gives an overview of a set of challenges addressed in this thesis. These can be summarized as follows:

- How to determine which model to choose?
  The field of learning has a long history in computer science [68]. As a result a wide range of techniques, such as linear models, neural network, regression trees or support vector machines, have been developed to tackle the problem of modeling the relationships existing in a set of data. The choice of a model that fits the data is however often difficult in practice, particularly when there is no a priori information on the relationships existing in the data. The limited processing resources of sensor nodes can furthermore exclude the use of computationally intensive learning techniques.

- How to compute the parameters of a model in a distributed way?
  When a model incorporates the spatial domain, this inevitably implies communication between sensor nodes in order to assess the relationships between the sensor measurements over space. The design of efficient distributed strategies to assess the spatial relationships is very challenging, and considerably reduces the range of techniques that can effectively be considered.

- How to assess the ability of a model to properly extract the information of interest?
  In practice, the accuracy of model can be assessed using different criteria, which depend on the application requirements. It can be for example the maximum deviation between the true measurements and the approximated measurements provided by the model, or the percentage of correct classification. An important issue in this respect consists in providing the observer with good estimates of the ability of the model to extract the requested information, and to ensure that this ability is maintained over time.

In the light of these challenges, the first main contribution of this thesis is the design of a model selection scheme suitable for low resource wireless sensor nodes. This scheme, called *Adaptive Model Selection* (AMS), is based on temporal modeling. It extends previous work by allowing sensor nodes to determine autonomously which model best fits their measurements, while guaranteeing that approximations are within an error bound $\pm\epsilon$ defined by the observer.

The second main contribution is the design of a distributed implementation of the principal component analysis (DPCA), a modeling method which allows to remove the correlations between sensor measurements. The method provides the following advantages. First, PCA provides varying levels of compression accuracies, ranging from constant approximations to full recovery of original data. It can therefore be used to trade application accuracy for communication costs, thus making the principal component aggregation scheme scalable with the network size. Second, the DPCA scheme demands all sensors to send exactly the same number of packets during each transmission, thereby balancing the communication costs among sensors. Given that communication costs is strongly related to the energy consumption, we also show that the balanced loading increases the network lifetime as well.

1.3 Outline of the thesis

Chapter 2 provides the reader with an overview of the notions and tools that will be considered in the thesis. The chapter is divided in two parts. The first deals with the domain of wireless sensor networks, and presents the technology, the applications, and the WSN characteristics, followed by an overview of the main networking frameworks. The second introduces the reader with the domain of supervised learning, and covers the statistical framework, the practical challenges, and the learning methodology.

Chapter 3 reviews the different modeling approaches which have been investigated in the literature to address the issue of energy efficient environmental monitoring. After introducing some notations and illustrative examples, the chapter is structured along three main parts, each of which focuses on a specific strategy, namely, the *model-driven data acquisition*, the *replicated model* approach, and the *aggregative* approaches.

The main contributions of this thesis, namely the AMS and the DPCA, are presented in Chapters 4 and 5, respectively. A short overview of these contributions is given below in Section 1.4. Chapters 4 is concluded by an experimental evaluation of the AMS on fourteen sets of measurements. The evaluation of the DPCA is the subject of chapter 6. The tradeoffs between accuracy and communication costs are analyzed using two data sets. The first data set is based on an illustrative scenario which simulates the appearance and disappearance of a set of patterns in an environment. The second data set consist of real-world temperature measurements taken by 52 sensors over a five day-period.

Chapter 7 summarizes the main results of this thesis, and identifies a set of future directions in the broader context of information processing and routing in wireless sensor networks. The remainder of the present chapter provides a detailed description of the thesis' contributions, and summarizes the notations that will be used throughout the following chapters.

1.4 Contributions

1.4.1 Adaptive Model Selection

The first main contribution of this thesis is the design of a model selection scheme for strategies based on replicated temporal models. Replicated temporal models have been among the first approaches investigated to reduce the data traffic by means of predictive models. Their rationale consists in using models able to predict the sensor measurements over time. Each sensor node computes its own model on the basis of the past measurements it has collected, and communicates it to the base station. The model is then used by the base station to provide the observer with approximations of the sensor measurements. The approximation error is bounded by an observer defined error threshold ϵ, which is known by all sensor nodes. Every time a new measurement is collected by a sensor node, the error between the model prediction and the measurement is computed. If the error is higher than ϵ, the node either transmits the measurements, or computes and communicates a new model. If the error is less than ϵ, then no message is sent, as the copy of the model running at the base station is known to provide an approximation that is within ϵ of the true measurement.

The ability of this approach to reduce the data traffic depends on the ability of the predictive model used to accurately predict the sensor measurements. Different strategies have been investigated in the literature, using different modeling techniques and methodologies to compute and update the models. The first contribution in this thesis is to give a close look at the employed methodologies, and to point out that simple models are in many cases the most likely to provide the highest communication savings. The reasons are twofold. First, complex models typically rely on more parameters, and therefore require more communication when a model is updated. Second, the use of high number of parameters raises estimation issues, which in practice decrease the accuracy of the predictive model.

This leads us to investigate a scheme based on model selection, called *Adaptive Model Selection* (AMS) [52], which aims at ensuring that the use of prediction models effectively decreases the amount of communication. It extends previous work in the domain by (i) allowing models of increasing complexity to be assessed in a competitive fashion, (ii) avoiding the use of models that increase the amount of communication, and (iii) providing an extensive empirical study which shows that in many cases, simple models have to be preferred over more complex models.

1.4.2 Distributed Principal Component Analysis

The second main contribution is the design of a distributed implementation of the principal component analysis (PCA). The PCA is a classic multivariate data processing technique that allows to reduce the dimensionality of the data [67]. More precisely, it provides a way to linearly transform a set of high dimensional data to a smaller space, in such a way that the approximation error is minimized. It is particularly efficient for correlated data, which is one of the main characteristics of sensor network data [9]. The purposes of using PCA are numerous. In particular, it allows to compress data, to filter noise, to extract feature of interest, or to detect unusual data patterns.

The principal component analysis consists of two steps. In a first stage, the basis vectors that characterize the subspace are computed. In a second stage, data are transformed from the original space to the subspace by projecting them on the basis vectors identified in the first stage. This process normally requires to centralize the data on a single computing unit.

In this thesis, we show that for sensor networks, part of the computation required for the PCA can be achieved in a distributed fashion [48]. More precisely, we first show that the projections on the PCA subspace can be computed along a routing tree. This technique called *Principal Component Aggregation* (PCAg), allows to carry out the computation of the projections within the network. We show that the PCAg can considerably reduce the amount of transmissions in the network as the whole set of measurements does not need to be retrieved at the base station.

Second, we investigate an approach allowing to compute approximations of the basis vectors of the PCA subspace in a distributed manner. In the PCA, the basis vectors are obtained by computing the main, or principal, eigenvectors of the covariance matrix of the measurements. We show that the power iteration method, a state-of-the-art technique for computing the eigenvectors of a matrix, can be implemented using a routing tree [49]. The use of the method requires the computation of the covariance matrix. We finally show that this computation is possible in a distributed manner, provided that the measurements of distant sensors are uncorrelated [51]. The resulting algorithm is called the *Distributed Computation of the Principal Components* (DCPC).

Additional contributions

Besides these main contributions, the work carried out during this thesis has led to the following other contributions.

Simulated data set with differential equations

When starting this thesis, very few real-world sensor deployments had been carried out, and thus there was a lack of data for assessing learning algorithms in different types of scenarios.

A first step in my work was therefore to produce simulated data sets, reflecting the types of measurements that could be captured by sensor networks. Joint work with Mehdi Moussaid led to investigate the use of partial differential equations, and in generating different data sets simulating diffusion and wave processes [50]. These data sets and implementation were made freely available to the research community by means of the website:

`http://www.ulb.ac.be/di/labo/`

Wireless sensor programming and real-world deployments

The acquisition of wireless sensor prototypes during the second year of my thesis allowed to carry out different network deployments and to collect real-world measurements in a variety of conditions. In each deployment, temperature, humidity and light measurements were collected. The deployments were carried out in the following environments:

- Cluster room in ULB NO building: Deployment of 7 sensors for one week. Measurements taken every 5 minutes.
- Library of the ULB Computer Science department: Deployment of 20 wireless sensors. Measurements taken every minute for one day.
- Experimental room of the ULB Service of Social Ecology (Pr. Jean-Louis Deneubourg). 20 sensors deployed for one day. Measurement taken every minute.
- Greenhouse of the ULB Solbosch campus: 18 sensors deployed over a 5 day period. Measurements taken every 5 minutes.

In order to provide the research community with the real-world data collected, these data sets were made publicly available by means of the website dedicated to WSN activities. Figure 1.3 gives some pictures and the sensor placement during the Solbosch greenhouse deployment. The code for the Adaptive Model Selection was written in TinyOS by Silvia Santini from the ETHZ, Zurich, Switzerland [78].

Localization and tracking

Localization and tracking techniques aim at estimating the position of a mobile object in an environment. Wireless sensor networks can be used for this task, by means of the radio module. Assuming that the mobile node can communicate with a set of fixed nodes whose positions are known, it is possible to estimate the distance separating the mobile nodes from the fixed nodes by means of the radio signal strength at the receiving nodes.

I took part in several experiments on that topic during my thesis. These experiments involved the deployment of sensors in different types of environments, in which the received signal strength of sensor nodes were logged over time. In particular, two deployments of twenty nodes were carried out in the library of the department (publicly available on the website), and a deployment of 52 sensors at the electricity plant of Drogenbos, Belgium, was also carried out as part of a localization project in industrial settings.

The data sets allowed to calibrate models for predicting the distance on the basis of the radio signal strength, and to assess the use of learning techniques such as model combination for improving the localization accuracy. This work was achieved as part of the PIMAN

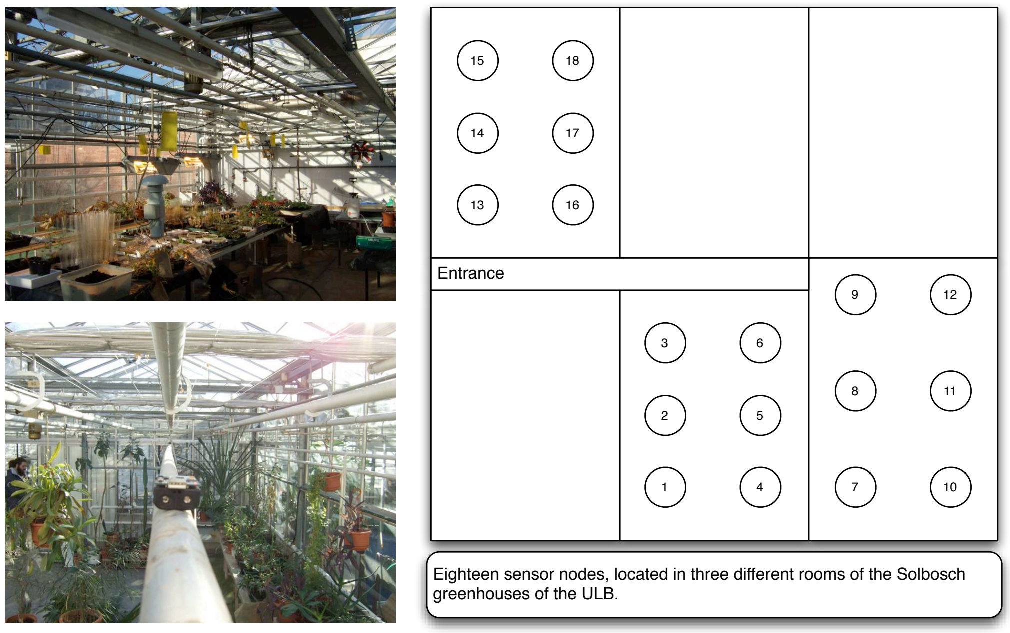

15
18
14
17
13
16
Entrance
9
12
3
6
8
11
2
5
1
4
7
10
Eighteen sensor nodes, located in three different rooms of the Solbosch greenhouses of the ULB.

Figure 1.3: Pictures and placements of sensor nodes during the Solbosch deployment.

project [1]. Our experiments led to the design of an architecture for tracking, which included sensor calibration, Kalman filtering, and motion detection by means of an accelerometer. These results were published in [21, 22].

1.5 Publications

The complete list of work published during this thesis is summarized below, by category and chronological order.

Book chapter

- Y. Le Borgne, J.M. Dricot, G. Bontempi. Principal Component Aggregation for Energy-efficient Information Extraction in Wireless Sensor Networks. Chapter 5, pages 55-80 in Knowledge Discovery from Sensor Data, Taylor and Francis/CRC Press, 2008.

Journals

- Y. Le Borgne, S. Raybaud, and G. Bontempi. Distributed Principal Component Analysis for Wireless Sensor Networks. Sensors Journal, 8(8):4821-4850, August 2008. MDPI.

[1]PIMAN - Pôle de compétence en Inspection et Maintenance Assistée par langage Naturel. Funded by the Région Bruxelles-Capitale (2007-2008).

- A. A. Miranda, Y. Le Borgne, and G. Bontempi. New Routes from Minimal Approximation Error to Principal Components. Neural Processing Letters, 27(3):197-207, June 2008. Springer

- Y. Le Borgne, S. Santini and G. Bontempi. Adaptive Model Selection for Time Series Prediction in Wireless Sensor Networks. Journal of Signal Processing, 87(12):3010-3020, December 2007. Elsevier.

Conferences

- J.M. Dricot, M. Van Der Haegen, Y. Le Borgne and G. Bontempi. A Modular Framework for User Localization and Tracking Using Machine Learning Techniques in Wireless Sensor Networks. Proceedings of the 8th IEEE Conference on Sensors, pages 1088-109. IEEE Press, Piscataway, NJ, 2008.

- J.M. Dricot, M. Van Der Haegen, Y. Le Borgne and G. Bontempi. Performance Evaluation of Machine Learning Technique for the Localization of Users in Wireless Sensor Networks. In L. Wehenkel and P. Geurts and R. Marée, Editors, Proceedings of the BENELEARN Machine Learning Conference, pages 93-94. 2008.

- Y. Le Borgne and G. Bontempi. Unsupervised and Supervised Compression with Principal Component Analysis in Wireless Sensor Networks. Proceedings of the Workshop on Knowledge Discovery from Data, 13th ACM International Conference on Knowledge Discovery and Data Mining, pages 94-103. ACM Press, NY, 2007.

- Y. Le Borgne, M. Moussaid, and G. Bontempi. Simulation architecture for data processing algorithms in wireless sensor networks. Proceedings of the 20th Conference on Advanced Information Networking and Applications (AINA), pages 383-387. IEEE Press, Piscataway, NJ, 2006.

- Y. Le Borgne, G. Bontempi. Round Robin Cycle for Predictions in Wireless Sensor Networks. Proceedings of the 2nd International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), pages 253-258. IEEE Press, Piscataway, NJ, 2005.

- G. Bontempi, Y. Le Borgne. An adaptive modular approach to the mining of sensor network data. Proceedings of the workshop on Data Mining in Sensor Networks. SIAM SDM, pages 3-9. SIAM Press, Philadelphia, PA, 2005.

Technical report

- Y. Le Borgne. Bias variance trade-off characterization in a classification. What differences with regression? Technical Report N°534, ULB, January 2005.

1.6 Notations

Throughout this thesis, boldface denotes random variables and normal font refers to the realizations of random variables. Lowercase letters denote scalars or vectors of scalar, and uppercase letters denote matrices.

Generic Notations

ϑ	Scalar or vector of scalars.
ϑ_j	j-th element of ϑ when ϑ is a vector.
$\boldsymbol{\vartheta}$	Random variable or random vector.
$\boldsymbol{\vartheta}_j$	j-th element of $\boldsymbol{\vartheta}$ when $\boldsymbol{\vartheta}$ is a vector.
$M_{[N\times n]}$	Matrix with N rows and n columns.
$M_{(i,j)}$	Element at the i-th row and j-th column of a matrix M.
$\mathbf{M}$	Random matrix.
M^T	Transpose of the Matrix M.
$\hat{\vartheta}$	Approximation of ϑ.

Probability and Stochastic Processes Theory Notations

Ξ	Set of possible outcomes.
ξ	Outcome (or elementary event).
Ω	Set of possible events.
ω	Event.
$\bar{\omega}$	Complementary of event ω.
$\mathcal{P}(\omega)$	Probability of the event ω.
$\mathbf{x}$	Random variable.
x	Realization of a random variable $\mathbf{x}$.
$p(x)$	Probability density for the random variable $\mathbf{x}$. Also $p_{\mathbf{x}}(\mathbf{x}=x)$ or $p_{\mathbf{x}}(x)$.
$u(.)$	Scalar field.
$\mathbf{x}(.)$	Stochastic process.
$\mathbf{x}[t]$	Discrete-time stochastic process.
$\boldsymbol{\nu}$	Random noise variable.
$\boldsymbol{\nu}[t]$	Random noise discrete-time process.
$\mu_{\mathbf{x}}$	Mean of a random variable $\mathbf{x}$.
$\Sigma_{\mathbf{x}}$	Covariance matrix of a random vector $\mathbf{x}$.
$\sigma_{\mathbf{x}}$	Variance of a random variable $\mathbf{x}$.

Learning Theory Notations

$\mathcal{L}$	Learning algorithm.
$\mathcal{X} \subset \mathbb{R}^n$	Input space.
$\mathbf{x} \in \mathcal{X}$	Input random variable.
$\mathcal{Y} \subset \mathbb{R}$	Output space.
$\mathbf{y} \in \mathcal{Y}$	Output random variable.
N	Number of available observations.
$x_{[i]} \in \mathcal{X}$	i-th observation in a set of observations.
$y_{[i]} \in \mathcal{Y}$	i-th observation in a set of observations.
$X_{[N\times n]}$	Matrix of input observations.
$Y_{[N\times 1]}$	Matrix of output observations.
$D_N = \{X, Y\}$	Training set.
$\Lambda \subset \mathbb{R}^p$	Model parameter space.
$\theta \in \Lambda$	Model parameter vector.
θ_j	j-th element of a model parameter vector.
Λ_θ	Class of models with parameters θ.
$h(x, \theta) \in \Lambda_\theta$	Prediction model with inputs x and parameters θ. Also $h_\theta(x)$, $h(x)$, or h.
$h_k(x, \theta)$	k-th model in a collection of K models. Also $h_k(x)$ or h_k.
$L(y, h_\theta(x))$	Loss function.
$E_{\text{emp}}(\theta)$	Empirical risk.
θ_{D_N}	$\arg\min E_{emp}(\theta)$.
G_N	Generalization error.
$w_k \in \mathbb{R}^n$	k-th principal component.
λ_k	k-th eigenvalue.
q	Number of retained principal components.
$W_{[n\times q]}$	Matrix of retained principal components.
$z \in \mathbb{R}^q$	Principal component scores.

Sensor Data Streams Notations

S	Number of sensors.
$t \in \mathbb{N}$	Time index.
$\mathcal{S}$	Set of sensors, identified by index i, $1 \leq i \leq \lvert S \rvert$.
$c_i \in \mathbb{R}^d$	Coordinates of sensor i.
$\mathbf{s}_i \in \mathbb{R}$	Random variable of measurements for sensor i.
$\mathbf{s} \in \mathbb{R}^S$	Random vector of measurements for sensors in $\mathcal{S}$
$p_{\mathbf{s}}(s)$	Joint probability density of the collected measurements.
$\mathbf{s}_i[t]$	Discrete-time stochastic process underlying sensor i measurements.
$s_i[t]$	Measurement of sensor i at time t.
$s[t] \in \mathbb{R}^S$	Vector of measurements of all the sensors at time t.
$X[N]$	Matrix of size $\times S$, containing N vectors of measurements $s[t]$ from time $t = 1$ up to time $t = N$.
$\mathcal{N}_i$	Neighborhood of sensor i, i.e., sensors that are within radio range of sensors i.
$i^*_{\mathcal{N}}$	Node i whose number of neihgbors is the highest.
$\mathcal{C}_i$	Set of children of sensor i in a tree topology.
$i^*_{\mathcal{C}}$	Node i whose number of children in the routing tree is the highest.
ϵ	Error threshold.
$\langle X \rangle$	Partial state record.
$\langle X \rangle = i(x)$	Initialization function.
$\langle Z \rangle = f(\langle X \rangle, \langle Y \rangle)$	Merging function.
z=$e(\langle X \rangle)$	Evaluator function.

Traffic load notations

All these quantities are in number of packets per epoch. HNL stands for highest network load.

Tx_i	Number of packets transmitted by sensor i.
Rx_i	Number of packets received by sensor i.
L_i	Network load of sensor i.
$L^{\mathrm{MD}}_{\mathrm{max}}$	HNL for the model-driven data acquisition approaches.
$L^{\mathrm{RM}}_{\mathrm{max}}$	HNL for the replicated models approaches.
$L^{\mathrm{DR}}_{\mathrm{max}}$	HNL for the distributed regression approaches.
$L^{\mathrm{D}}_{\mathrm{max}}$	HNL for a D action.
$L^{\mathrm{A}}_{\mathrm{max}}$	HNL for an A action.
$L^{\mathrm{F}}_{\mathrm{max}}$	HNL for an F action.
$L^{\mathrm{Cov_{cent}}}_{\mathrm{max}}$	HNL for the centralized estimation of the covariance matrix.
$L^{\mathrm{Cov_{dist}}}_{\mathrm{max}}$	HNL for the distributed estimation of the covariance matrix.
$L^{\mathrm{EV_{cent}}}_{\mathrm{max}}$	HNL for the centralized eigendecomposition.
$L^{\mathrm{EV_{dist}}}_{\mathrm{max}}$	HNL for the distributed eigendecomposition.

Abbrevations and acronyms

AMS	Adaptive Model Selection.
AR(p)	Autoregressive model of order p.
AS	Aggregation Service.
COTS	Components-off-the-shelf.
DPCA	Distributed Principal Component Analysis.
DCPC	Distributed Computation of the Principal Components.
GPS	Geopositioning System.
HNL	Highest network load.
LMS	Least Mean Square.
MAC	Medium Access Control.
PCA	Principal Component Analysis.
PIM	Power Iteration Method.
RM	Replicated model.
SNR	Signal-to-Noise Ratio.
TAG	Tiny Aggregation.
TDMA	Time Division Multiple Access.
CSMA	Carrier Sense Multiple Access.
WSN	Wireless Sensor Networks.

Chapter 2

Preliminaries

In this chapter, Section 2.1 gives a background on the challenges raised by WSN, together with the strategies that can be used at the network level to improve the energy-efficiency of environmental monitoring applications. Section 2.2 covers the basics of supervised learning, which will be used in this thesis to model sensor network data in order to reduce the amount of communication in a WSN.

2.1 Wireless sensor networks

Wireless sensor networks differ from other networks in a number of ways. This section provides a state-of-the-art on the use of WSN for environmental monitoring, and is structured as follows. Section 2.1.1 provides an overview of the current technology, and illustrates the constraints of wireless sensors in terms of computational, network throughput and energy resources. Section 2.1.2 describes the main actors of environmental monitoring with WSN, and introduces the issues related to networking and interfacing WSN. Section 2.1.3 emphasizes one of the main characteristics of WSN, called *data-centric networking.* Section 2.1.4 finally presents optimization strategies which can be used to reduce energy consumption in environmental monitoring tasks, and in particular describes *aggregation services.*

2.1.1 Sensor technology

A wireless sensor, or sensor node, is a device typically composed of a microprocessor, a memory, a radio transceiver, a power source and one or more sensors [3]. In many WSN applications, it is important that the measurements are geolocalized. When sensors are randomly deployed, sensor nodes may also feature a geopositioning system (GPS) to obtain the location information. The schematic of a basic wireless sensor network devices is represented in Figure 2.1.

The current generation of commercially available wireless sensor hardware have the size of a small wallet, and are designed mainly for experimental research purposes. The next generation of wireless sensors is well illustrated by the seminal *smart dust* project [90], which took place at the University of Berkeley between 1998 and 2001, and which led to the design of a laboratory prototype wireless sensor whose volume was about 5mm^3. The design of viable millimeter scale wireless sensors is however still the subject of research efforts. The resources available on a sensor node inevitably depend on its size, and the smaller the

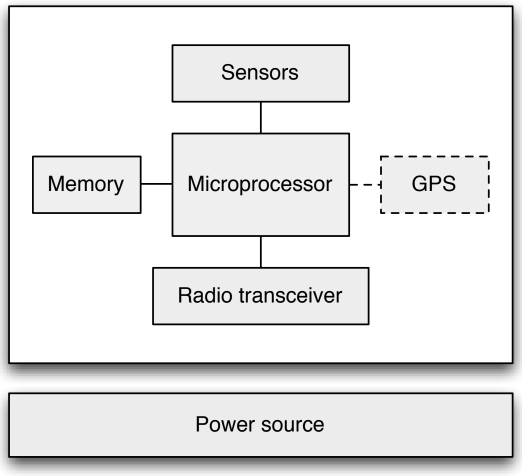


Figure 2.1: Schematic of a basic wireless sensor network device [45].

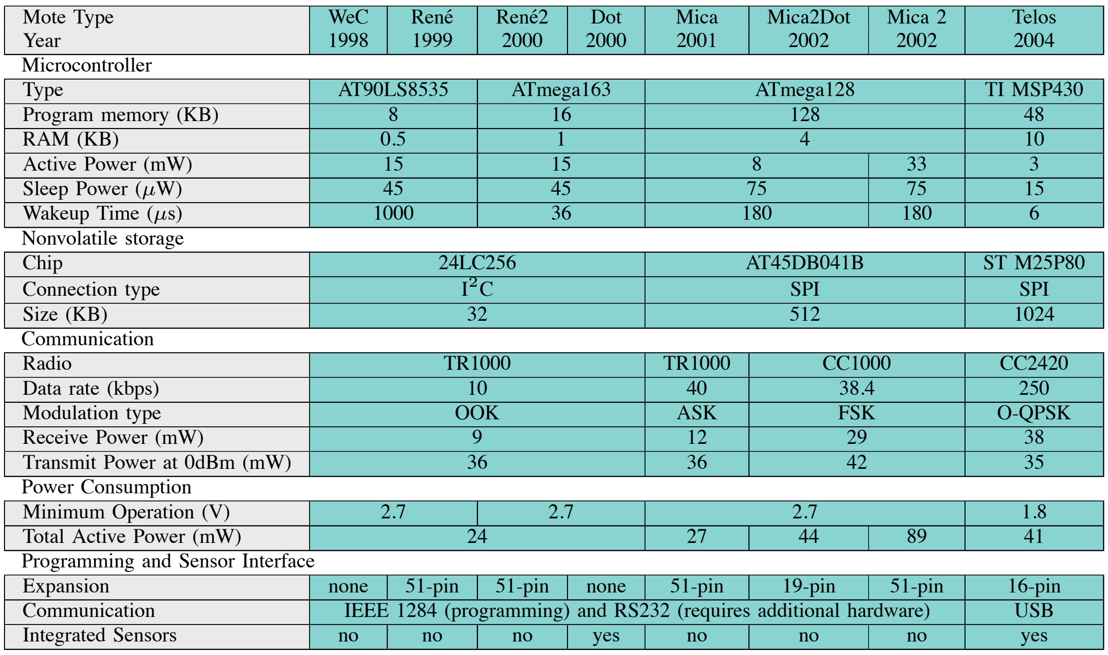
volume, the lesser the computational, memory, bandwidth and energy resources available. Table 2.1 compares the resources available on sensor nodes as their size decreases.

	WINS NG 2.0	MicaDot	Smart Dust
Cost	\$100s	\$10s	<\$1
Size (cm^3)	5300	40	.005
Weight	5400	70	.002
Battery capacity	300	15	(Less)
Sensors	Off-board	Integrated on PCB	MEMS
Memory	32MB RAM	4KB RAM	(Less)
CPU	400 MIPS	4 MIPS	(Less)
Radio range	100m	30m	(Less)
Operating system	Linux	TinyOS	(Smaller)

Table 2.1: Comparison of the resources available on sensor nodes as the size decreases [95].

The need for experimenting WSN protocols, algorithms and applications in real-world conditions has led to the design of wireless sensor platforms for research. The University of Berkeley initiated such designs in 1999 with the WesC, a platform that approximated the functionalities envisioned by the Smart Dust project. This design was followed by the MICA, MICAz, MICA2DOT, and TelosB platforms, which have been the most widely used sensor nodes in academia for prototypical WSN deployments. The characteristics of these sensor nodes, also known as *motes*, are summarized in Figure 2.2.

These nodes use components-off-the-shelf (COTS) hardware instead of integrated silicon designs in order to allow easy customization of the boards, and to reduce the production costs. Following the Smart Dust vision, these platforms run 8-bit microcontrollers and have a few tens of kilobytes of memory. Low data rate radios, ranging from a few tens to a few hundreds of kilobits per second, enable the communication. Sensors are either integrated



Mote Type	WeC	René	René2	Dot	Mica	Mica2Dot	Mica 2	Telos
Year	1998	1999	2000	2000	2001	2002	2002	2004
Microcontroller								
Type	AT90LS8535		ATmega163		ATmega128			TI MSP430
Program memory (KB)	8		16		128			48
RAM (KB)	0.5		1		4			10
Active Power (mW)	15		15		8		33	3
Sleep Power (μW)	45		45		75		75	15
Wakeup Time (μs)	1000		36		180		180	6
Nonvolatile storage								
Chip	24LC256				AT45DB041B			ST M25P80
Connection type	I^2C				SPI			SPI
Size (KB)	32				512			1024
Communication								
Radio	TR1000				TR1000	CC1000		CC2420
Data rate (kbps)	10				40	38.4		250
Modulation type	OOK				ASK	FSK		O-QPSK
Receive Power (mW)	9				12	29		38
Transmit Power at 0dBm (mW)	36				36	42		35
Power Consumption								
Minimum Operation (V)	2.7		2.7		2.7			1.8
Total Active Power (mW)	24				27	44	89	41
Programming and Sensor Interface								
Expansion	none	51-pin	51-pin	none	51-pin	19-pin	51-pin	16-pin
Communication	IEEE 1284 (programming) and RS232 (requires additional hardware)							USB
Integrated Sensors	no	no	no	yes	no	no	no	yes

Figure 2.2: The family of Berkeley motes and their capabilities [74].

(such as on the dot 2000 or the TelosB), or attached by means of a daughter board. Their size essentially depend on the batteries, typically a pair of AA cells. The integration of these COTS platforms in silicon would reduce their size to a few millimeter cube, as illustrated by the Spec platform, the silicon integrated counterpart of the MICA platform in the Smart Dust project. Such small scale wireless sensors are however still not viable, as a number of issues in hardware robustness and communication protocols must be further investigated [90].

Among current platforms, the MICA motes and the Telos are nowadays the most popular ones for WSN prototyping. The Telos outperforms the MICA motes, particularly in terms of radio throughput and power consumption. The Telos architecture features lower power electronics for the flash memory and the microprocessor, and the radio rate increased from about 40 kbps to 250 kbps. Despite these advantages, the earlier appearance of the MICA motes family on the market, and the relatively small differences in functionalities between the MICA motes and the TelosB led a large number of laboratories to adopt the MICA motes for their experimental work.

2.1.2 Environmental monitoring

The focus of this thesis is data collection for environmental monitoring, where the task consists in retrieving the measurements from the sensor network at regular time intervals. In a typical environmental monitoring scenario, four main entities interact [84]:

1. *The phenomenon*: It is the entity of interest to monitor. The phenomenon may be the vibration patterns on a bridge, the location and size of a fire in a forest, or the number

and type of animals present in an environment. The data collected about the phenomenon may be analyzed/filtered by the sensor network before being communicated to the base station/ the observer.

2. *The observer*: It is the end user or the application interested in obtaining information collected by the sensor network. The observer may formulate queries to the network and receive responses to these queries by means of the *base station*.

3. *The sensor nodes, or sensors*: They are the wireless devices that implement the physical sensing of an environment and the reporting of the measurements to a base station. The sensors may be placed one by one in the sensor field, according to predetermined positions, or randomly deployed, from an airplane for example. Once deployed, sensors are assumed to be static, and to run unattended with a non renewable amount of energy.

4. *The base station*: It is assumed to possess higher resources than sensor nodes, and provides a centralized data storage and processing unit for the data collected by the sensor network. It may connect the WSN to the Internet.

The design of data collection systems for sensor networks has been addressed early in the research literature [38, 2]. In particular, the three following aspects are of significant practical importance. First, an observer must be able to communicate with the network in order to retrieve data. In data collection tasks, it is in particular desirable that the user can select what measurements to retrieve, change the sampling frequency, or address a query to a specific region of the network. Second, once a query has been formulated by the observer, it must be communicated to the network, and sent to the appropriate nodes. Finally, once all the sensor nodes involved in the query have been reached, the query must be executed in such a way that the observer retrieves the requested information.

The collection of data from the sensors to the base station requires in most cases the establishment of a routing structure. The radio range of sensors is limited, and therefore the data from sensors distant to the base station must be relayed by intermediate sensors. The transmission of data using multiple relay nodes is called *multi-hop* routing. In data collection tasks, a typical routing structure is a tree connecting all the nodes to the base station [13, 56]. An example of such a tree, also known as data gathering tree, is given in Figure 2.3. The dotted circle illustrates the radio range of the dark gray sensor. The light gray sensors lying inside the circle form its *neighborhood*, i.e., the set of other sensors it can communicate with.

2.1.3 Data-centric paradigm

At the conceptual level, one of the fundamental differences between wireless sensor networks and other networks is that the routing and querying techniques can be made more efficient if the communication is based directly on application specific data content instead of the traditional IP-style addressing [45]. This focus on data content is referred to as the *data-centric* paradigm in [33], whose main characteristic is the routing based on attributes.

In attribute-based routing, the sensor nodes may be identified not on the basis of a network address, but on the basis of their *attributes*, i.e., the pieces of information they hold. These attributes include for example the location or the measurements of a sensor node. A reference approach that uses this feature is the Directed Diffusion technique [38]. In Directed

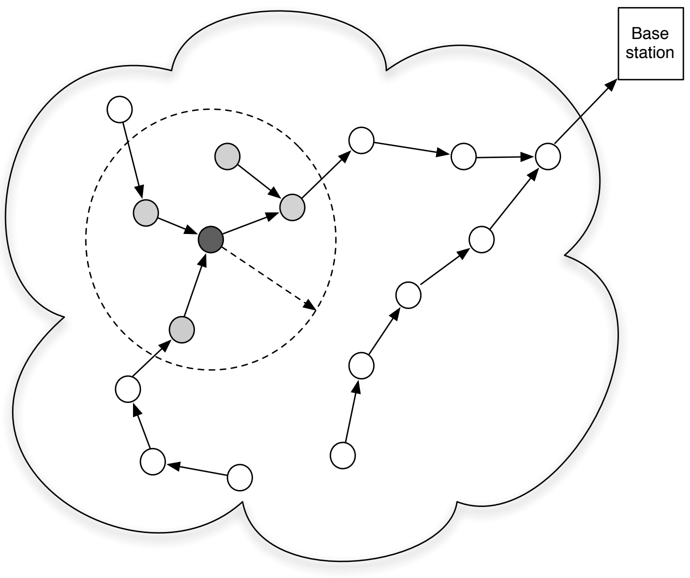


Figure 2.3: Data gathering tree enabling communication between the sensor nodes to the base station.

Diffusion, both the observer's queries and sensor's attributes are described through sets of *attribute-value* pairs. Thus, an observer may request sensor nodes within a given region to report their temperature measurements every thirty seconds for one hour by defining the set following attribute-value pairs:

Type=temperature	//Type of measurements required
Location=[0,0, 15, 35]	//Coordinates of the region of interest
Duration=10:00:00	//Duration of the query
Frequency=30	//Sampling frequency

The query is sent throughout the network, and executed by all sensor nodes that are located within the region and that have a temperature sensor. An example of response to the query can be:

Id=324	// Node unique identifier
Type=temperature	//named record type
Value=25.3°C	//value of this type
Location=[10,25]	//location of the measurement
Time=04:23:30	//time of measurements

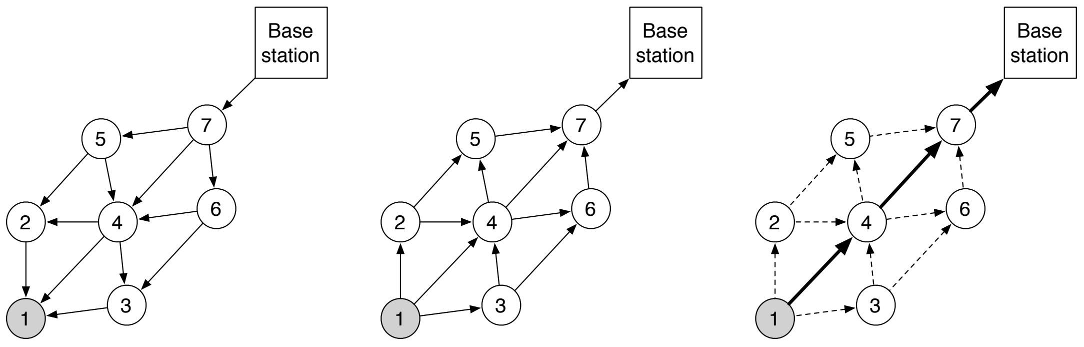
The nodes may have identifiers, but these do not require to be part of the observer's request. Directed diffusion is organized in three stages (Figure 2.4):

1. Query dissemination: When the observer is interested in collecting data, it sends

through the base station a query that specifies the type of data required. Each node, on receiving the query, rebroadcasts it to its neighbors. Figure 2.4(a) illustrates this stage, with node 7 transmitting to nodes 6 and 5, which themselves reach nodes 4, 2 and 3, up to node 1, which matches the query.

2. In addition, as the query is disseminated, each node sets up an *interest gradient* to the nodes from which the query was received. The gradient quantifies how efficient the different nodes are at routing the query. The way the gradient is computed is application specific, and aims at minimizing a certain cost, such as the number of hops or the traffic load on a given path.

3. Data are then transmitted from node 1 to the base station using the path that has the best gradients. Multipath delivery can be used in order to make the message delivery more robust to transmission failure.



(a) Query dissemination. (b) Gradients setup. (c) Data delivery along the reinforced path.

Figure 2.4: Main stages of Directed Diffusion [38]. The sensor node that matches the query is node 1.

The use of attributes to match sensor nodes and observer's interests make sensor networks conceptually close to databases. In classical database management systems, data is accessed by queries from users or applications which specify information to retrieve in a high level language such as SQL. The SQL syntax has therefore been introduced in the WSN domain very early, and has been extended to meet some of the specificities of WSN, in particular by adding the clauses related to the frequency and duration of the data collection [56, 57, 27]. Using SQL, the query formulated above in terms of attribute-value pairs becomes

```
SELECT temperature FROM sensors
WHERE location=[0,0, 15, 35]
DURATION=00:00:00,10:00:00
EPOCH DURATION 30s
```

The fact that the SQL syntax is widely known thus makes the interfacing between observers and sensor networks more natural and simple.

2.1.4 Energy efficiency and aggregation services

Among the sensor nodes resources, energy is widely considered as the most precious resource as it determines the lifetime of an application. In full active mode, the lifetime of a sensor node such as a Tmote is only of a few days. Improvements in battery design and energy harvesting techniques only offer partial solutions to extend sensor nodes' lifetime, and therefore most of the WSN literature has focused on the design of protocols with energy efficiency as the primary goal [45].

Energy consumption depends on the amount and type of activities performed by sensor nodes. In order to more precisely control energy consumption, sensor nodes are usually designed so that their different components can be powered on and off. The current consumption is the lowest in the standby mode, in which all components are switched off except the clock, and is on the order of a few microampers. The use of the microcontroller unit (MCU) typically requires a few milliampers, and is therefore three orders of magnitude higher than the standby mode. The additional use of the radio or the memory component increases by an order of magnitude the overall current draw [73]. This is illustrated in Table 2.2, which details the current draws of the MICA2, MICAZ and Telos motes for different modes of operations.

Operation mode	MICA2	MicaZ	Telos
Standby	19.0 μA	27.0 μA	5.1 μA
MCU Idle	3.2 mA	3.2 mA	54.5μA
MCU Active	8.0 mA	8.0mA	1.8 mA
MCU + Radio RX	15.1 mA	23.3 mA	21.8 mA
MCU + Radio TX (0dBm)	25.4 mA	21.0 mA	19.5 mA
MCU + Flash Read	9.6 mA	9.4 mA	4.1 mA
MCU + Flash write	21.6 mA	21.6 mA	15.1 mA

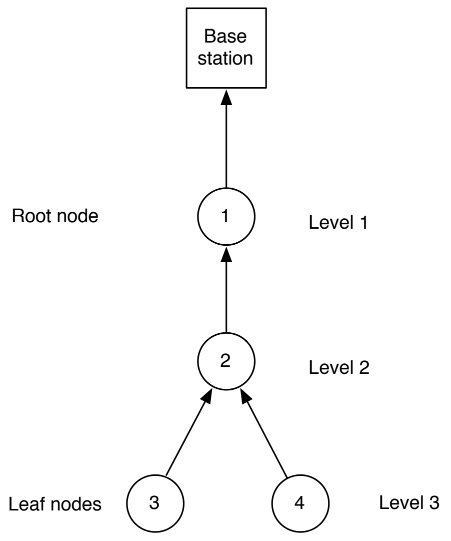
Table 2.2: Comparison of the current consumption of MICA2, MICAZ and Telos mote [73].

A simple solution to save energy and extend the lifetime of a sensor node is to operate the node with periodic switching between standby and active modes. This approach, called *duty-cycling*, is particularly suitable for periodic data collection tasks [3, 45, 83], where the operation of the sensors is by definition periodic. Since the collection and the transmission of one packet is typically less than one second, the energy savings can be significant if the frequency of the data collection is low. The sampling of one measurement every minute can for example extend the lifetime of a node by a factor of at least sixty using the duty-cycling.

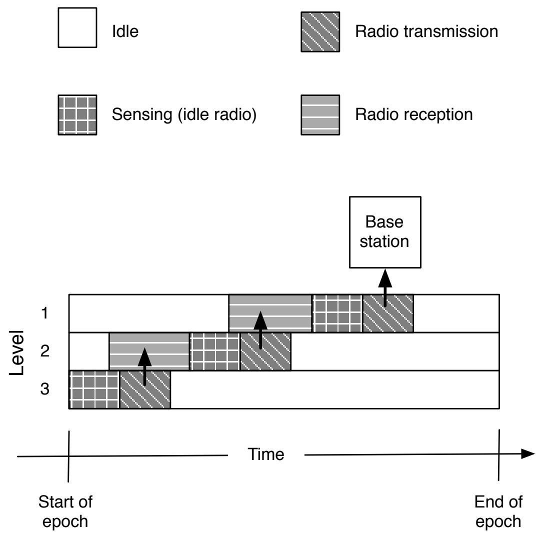
In multi-hop networks, the nodes must be synchronized so that parents and children in the routing tree are active at the same moment. Different systems, such as TAG [56, 58], Cougar [93] or Dozer [13] for example, have been proposed to carry out the synchronization in a time and energy-efficient way. TAG is the most well-known representative of such systems.

TAG stands for Tiny AGgregation and is an aggregation service for sensor networks which has been implemented in TinyOS [85], an operating system with a low memory footprint specifically designed for wireless sensors. TAG allows to aggregate data within a WSN in a time and energy-efficient manner. To that end, an epoch is divided into time slots, in such a way that the activities of the sensors are synchronized as a function of their depth in the routing tree. Any algorithm can be used to design the routing tree, as long as (i) it allows the data to flow in both directions of the tree, and (ii) it avoids sending duplicates [56].

The goal of TAG is to minimize the amount of time spent by sensors in powering their different components and to maximize the time spent in the standby mode, in which all electronical components are switched off. This synchronization allows to significantly extend the lifetime of the sensors. An illustration of the activities of the sensors during an epoch is given in Fig. 6.7, for a network of four nodes with a routing tree of depth three.



(a) Routing tree of depth three.



(b) Activities carried out by sensors depending on their level in the routing tree *(adapted from [56])*.

Figure 2.5: Multi-hop routing along a routing tree, and node synchronization for an efficient use of energy resources.

The establishment of the routing tree works in a similar manner as Directed Diffusion described in Section 2.1.3. The observer's query is flooded from the base station to the sensor nodes, which identify at the same time a possible routing tree. Each node chooses another node as its parent, using a metric such as the minimum number of hops. Nodes also synchronize their clocks by adding timestamps in the messages, and define their own activity schedule such that their radio transmission period is within the radio reception period of their parents. In order to overcome limitations in the quality of the clock synchronization between parents and children, the parents listen for longer than the transmission interval of their children. The duration of the interval in which parents receive the values from their children needs to be long enough so that all children can report, but not so long so that nodes deep in the tree can report their value before the end of the epoch.

Aggregation services such as TAG allow to both reduce energy consumption by carefully scheduling sensor node's activity, and by allowing the measurements to be aggregated as they are routed to the base station. This aggregation allows to reduce the number of packet transmissions, and can further be used to compute parameters in a distributed manner as will be discussed in more details in Section 3.

2.1.5 Summary

Wireless sensor networks are strongly resource constrained, particularly in terms of energy. In environmental monitoring applications, where the network is expected to run unattended for months or even years, the need for energy-efficient data collection schemes has driven the design of new routing and querying strategies. In particular, the optimization problems resulting from maximizing the sleeping time of sensor nodes has led to the design of aggregation services.

The strategies presented in this section however did not take into account the fact that sensor network data are often correlated over space and time. Given that the microprocessor consumes one order of magnitude less energy than the radio module, in-network processing strategies can de designed to detect these correlations, and to allow further energy savings by removing the correlations within the network. Learning techniques are in this respect among the most promising approaches.

2.2 Supervised learning

This section provides the reader with the main concepts and methods used in supervised learning. It introduces the general problem of learning with computers, and provides the general statistical framework that formalizes the actors of supervised learning tasks. The reader is assumed to be acquainted with the basics of statistical theory. Appendix A otherwise provides a review of the main statistical concepts.

Learning "by heart" is probably one of the easiest thing for a computer. However, in most cases, the goal pursued is not to make the computer store all the examples provided, but rather to make it "understand" the underlying relationship that exists between the training examples provided. In particular, the goal of a learning procedure is most often to make good predictions, i.e., to properly guess the relationships existing between data other than those used as examples. This ability, called *generalization*, raises particularly tough challenges.

Section 2.2.1 provides insights into the nature of these challenges by means of the bias/variance decomposition, and makes clear that in most cases, simple models are preferable to complex models. Section 2.2.2 addresses how a learning procedure can be designed to find a representation that suits the training examples, while allowing good generalization. Section 2.2.3 then presents different classes of models, with an emphasis on the class of linear models. Finally, the challenges raised by generalization may also be addressed by preprocessing the data, with the use of techniques called dimensionality reduction. Section 2.2.4 provides an overview of these techniques.

2.2.1 Learning from data

Learning with computers

Conceptually, learning can be defined as acquiring knowledge through experience. A definition that adapts the concept of learning to the field of computer science is proposed in [68]:

> A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

This definition is broad enough to encompass different sort of tasks, which would in the common sense be considered as learning tasks. Such tasks include for example the learning of a language, the forecasting of a situation based on previous observations, and the design of strategies for winning a game.

The subfield of computer science that deals with the design of algorithms and techniques that allow computers to learn is referred to as *machine learning*. Three types of learning tasks can be distinguished [23]. First, unsupervised learning, which deals with the design of strategies aimed at finding structures in data, but where no "supervisor" can say if the data structures obtained are correct. An example of such a task is the design of taxonomies, such as with species, where the performance can be assessed in terms of qualitative judgments on the clarity of the taxonomy. This judgment may evolve with experience as new species are discovered.

Second, reinforcement learning, which refers to learning tasks where the optimal solution is not known, but where some notion of reward can be used to orient the search in the space of solutions. The design of strategies in reinforcement learning problems often involves trials and errors. The development of strategies for winning a game or for finding the shortest path are examples of reinforcement learning tasks. Research as a whole also falls in that category.

Finally, supervised learning, which deals with the more "classic" notion of learning, where a supervisor, or teacher, can provide examples about some kind of relationships to a learner. The task is for the learner to uncover the relationships that can be found in the examples. In the terminology of machine learning, the examples are provided by a set of observations called the *training data* or *training set*. It is composed of pairs of *input* variables, typically vectors, and one or more *output* variables, which are assumed to be dependent on the inputs. The nature of the output variable can be categorical or continuous. In the former case, the supervised learning task is referred to as classification, and in the latter case, as regression. Classification is typically used for recognition tasks, such as character, image, or sound recognition. Applications of regression include for example interpolation, approximation, and prediction tasks.

Statistical definition of a supervised learning task

Classification and regression have a lot in common, and in particular both can be viewed as a task in function approximation [32]. Let $x \in \mathcal{X}$ denote the input variable, taking its values in an input domain $\mathcal{X}$, and $y \in \mathcal{Y}$ denote the output variable, taking its values in an output domain $\mathcal{Y}$. In the case of regression, the output space is usually the domain of real numbers $\mathbb{R}$. In the case of classification, it is usually of discrete set of elements, not necessarily numeric nor ordered.

In this thesis, we will be mainly concerned with regression tasks, and input and output domains will unless otherwise stated refer to the $\mathbb{R}^n$ and $\mathbb{R}$. We will occasionally consider classification tasks, in which case the nature of the output space $\mathcal{Y}$ will be specified.

Example 2.1: *Let us consider a sensor network collecting temperature measurements in an outdoor environment, two sensors close to each other are likely to collect similar measurements. The problem of estimating the measurements of one sensor given the other is a* ***regression*** *problem. The input is the measurement of one sensor, and the output the measurement of the other sensor. Inputs and outputs both belong to the set of real numbers.*

Example 2.2: *Let us consider a sensor network of n inclination sensors placed on the body of an individual, and let us assume that the goal is to determine, on the basis of inclination measurements, whether the individual is lying, sitting or standing. This is a* ***classification*** *problem. The input is the set of n sensor measurements, taking their values in n-dimensional space of real numbers, and the output the set of the three possible outcomes {lying,sitting,standing}.*

The main actors of a supervised learning task are the following:

- A *target operator* $\mathbf{y}(x)$, which represents the relationship that links inputs x to the output y. In the most general case, the relationship is unknown and non deterministic. The target operator is then a stochastic process, and the distribution followed by $\mathbf{y}(x)$ for a given x can be represented by a conditional distribution $p_{\mathbf{y}}(y|x)$. A joint distribution $p_{\mathbf{x},\mathbf{y}}(x, y)$ may equivalently be used to represent the stochastic relationship.

- A *training set* of N observations, or examples, of the relationship between the inputs and the outputs. It is denoted $D_N = \{(x_{[i]}, y_{[i]})\}$, where the pair $(x_{[i]}, y_{[i]})$ represents the i-th example. Each example is a sample of the joint distribution $p_{\mathbf{x},\mathbf{y}}(x, y)$.

- A *prediction model*, whose role is to represent the unknown relationship $\mathbf{y}(x)$ by means of a parametric function

$$\begin{aligned} h_\theta : \mathcal{X} &\rightarrow \mathcal{Y} \\ x &\mapsto \hat{y} = h_\theta(x) \end{aligned}$$

which, given an input x and a set of parameters $\theta \in \Lambda$, produces a prediction $\hat{y}$. The prediction model is assumed to be entirely defined by the set of parameters. In this thesis, the parameters space will in most cases be a p-dimensional domain of real numbers. Depending on the context, the notation $h(x, \theta)$ will also be used to make more explicit the dependence of h on θ. The prediction model is in this case seen as a function

$$\begin{aligned} h : \mathcal{X} \times \Lambda &\rightarrow \mathcal{Y} \\ x, \theta &\mapsto \hat{y} = h(x, \theta) \end{aligned}$$

where the parameters are also considered as variables of the prediction model.

- A *loss function*, defined as

$$\begin{aligned} L : \mathcal{Y} \times \mathcal{Y} &\rightarrow \mathbb{R} \\ y, \hat{y} &\mapsto L(y, \hat{y}) \end{aligned}$$

which quantifies the discrepancy between the actual output y and the predicted output $\hat{y}$. Loss functions depend on the type of task, in particular regression or classification. In the former case, the most widely used loss function is the *quadratic* loss function, which is defined as

$$\begin{aligned} L_2 : \mathcal{Y} \times \mathcal{Y} &\rightarrow \mathbb{R} \\ y, \hat{y} &\mapsto (y - \hat{y})^2. \end{aligned} \tag{2.1}$$

The function quantifies the euclidean distance between the predicted and the desired output. The popularity of this loss function in regression settings mainly stems from the fact that it is differentiable, which considerably simplifies optimization procedures related to the parametric identification of a prediction model (see Section 2.2.2 and 2.2.3).

In classification problems, the notion of euclidean distance is not appropriate for assessing the loss. The output space is a finite set of classes, not necessarily ordered. The most popular loss function, called the misclassification function, is defined as

$$\begin{aligned} L_{0/1} : \mathcal{Y} \times \mathcal{Y} &\rightarrow \{0, 1\} \\ y, \hat{y} &\mapsto \begin{cases} 0 & \text{if y} \neq \hat{\text{y}} \\ 1 & \text{if y} = \hat{\text{y}} \end{cases} \end{aligned} \tag{2.2}$$

and quantifies the match between the predicted and the desired output by a binary indicator.

- A *learning procedure* which consists, on the basis of a training set, in producing a model $h_\theta(x) = h(x, \theta)$ with $\theta \in \Lambda$. The goal is to minimize the loss between the prediction model outputs $\hat{y}$ and the actual outputs y. Assuming that the parameters θ completely define the prediction model, the parameter space Λ is also referred to as the *model space*. The learning procedure is an essential part of a learning task, and is the subject of Section 2.2.2.

An overview of the supervised learning setting is given in Figure 2.6.

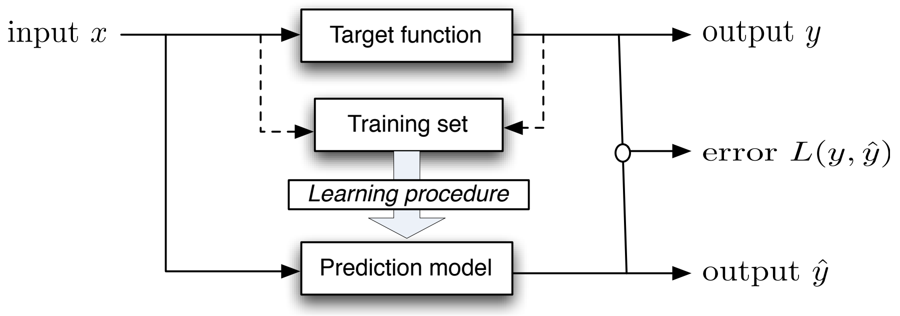


Figure 2.6: The supervised learning setting. A training set made of input and output observations of the target operator is used by a learning procedure to identify a prediction model of the target operator. The goal is to minimize the error between the true output y and the prediction model $\hat{y}$, quantified by a loss function $L(y, \hat{y})$.

Example 2.3: *Let us assume that the true relationship between x and y is defined by the stochastic process $\mathbf{y}(x) = x^2 + \boldsymbol{\nu}$ where $\boldsymbol{\nu} \sim \mathcal{N}(0, \sigma_{\boldsymbol{\nu}})$ is a Gaussian random noise, and that a training set of $N = 100$ examples has been collected. A linear model of the form $\hat{y} = \theta_1 + \theta_2 x$, although not optimal, could be used as a first try to represent the relationship. The parameter space is $\mathbb{R}^2$, the vector of parameters is $\theta = (\theta_1, \theta_2)$. The least square technique, based on the quadratic loss function, can be used as the learning algorithm to estimate the parameters θ. This technique will be detailed in Section 2.2.3.*

In qualitative terms, the goal of the learning procedure is to provide the best representation of the stochastic relationship $\mathbf{y}(x)$. The loss function allows to quantify the notion of best approximation. It is defined as the representation that minimizes the loss over the $\mathcal{X} \times \mathcal{Y}$ domain:

$$\begin{array}{rcl} f^* : \mathcal{X} & \rightarrow & \mathcal{Y} \\ x & \mapsto & y^* = \arg\min_{y'} E_{\mathbf{y}}[L(\mathbf{y}(x), y')]. \end{array}$$

This function will be referred to as the *target function.* The loss over the $\mathcal{X} \times \mathcal{Y}$ domain is the *functional risk*, defined for a function $f(x)$ as

$$R_f = E_{\mathbf{x},\mathbf{y}}[L(y, f(x))]$$

For $f = f^*$, the resulting loss is called the *minimal functional risk*, and is the expectation value $E_{\mathbf{x},\mathbf{y}}[L(y, f^*(x)]$. For a model $h_\theta(x)$, the *functional risk* can be defined as a function of the parameters $R(\theta) = E_{\mathbf{x},\mathbf{y}}[L(y, h_\theta(x))]$. Denoting by Λ^* a parameter space such that $\exists \theta^* \in \Lambda^*$ *s.t.* $\forall x, h_{\theta^*}(x) = f^*(x)$, the learning problem may then be stated as finding the model with parameters $\theta^* \in \Lambda^*$ such that

$$\theta^* = \arg\min_{\theta} E_{\mathbf{x},\mathbf{y}}[L(y, h(x, \theta))]$$

Such a model will be referred to as an *optimal model* h^*, as it verifies

$$\begin{array}{rcl} h_{\theta^*} : \mathcal{X} & \rightarrow & \mathcal{Y} \\ x & \mapsto & y^* = \arg\min_{y'} E_{\mathbf{y}}[L(\mathbf{y}(x), y')] \end{array}$$

whose loss is also the minimal functional risk. In most cases, there is very few assumptions, if any, that can be made about the analytical form of the target function. Models are defined by parameters, and in general, the higher the number of parameters, the greater the set of functions that can be represented. This calls for the use of complex models with high number of parameters, so that chances to properly represent the unknown target function are increased.

However, increasing the size of a parameter space leads to the two following practical issues. First, the parameter space may become so large that it is computationally intractable to find the right parameters, even if an unbounded number of examples is provided by the training set. Second, the size of the training set is in most cases fixed, and the number of examples at hand is too low to allow an effective exploration of the parameter space.

Thus, the choice of a model for a learning task is subject to the following tradeoff: on one hand, if the model is too simple, it will not be able to properly represent the target function. This outcome is referred to as as *underfitting.* On the other hand, if the model is too complex, it will be able to fit exactly the examples available, without finding an effective way of modeling the target operator. This outcome is referred to as *overfitting.*

A fruitful way to get insight into this tradeoff is provided by the noise, bias, and variance decomposition.

The noise, bias and variance decomposition

Given a training set D_N and a model $h(x, \theta)$, a learning algorithm aims at finding the parameters θ so that the loss on the training set is minimized. In the following, the dependence of the parameters θ on the training set is made explicit by the notation θ_{D_N}. The training set is the result of a random sampling of the distribution $p_{\mathbf{x},\mathbf{y}}(x, y)$, and therefore the training set is also a random variable.

In order to assess the average loss of a prediction model independently of a specific sampling D_N, much insight into learning algorithms is gained by raising the following question:

What is the influence of the training set on the overall loss of a prediction model?

This loss, called the *generalization error*, consists in quantifying the loss incurred by a learning algorithm over the $\mathcal{X} \times \mathcal{Y}$ domain, for all possible training sets of size N. Denoting by G_N this loss, we have

$$G_N = E_{\mathbf{x},\mathbf{y},\mathbf{D_N}}[L(y, h(x, \theta_{D_N}))].$$

The training set is here considered as a random variable, which allows to assess the learning algorithm independently of a particular training set. In the same manner, the generalization error for a given input x is defined by

$$G_N(x) = E_{\mathbf{y},\mathbf{D_N}}[L(y, h(x, \theta_{D_N}))].$$

In the case of the quadratic loss function, it can be shown that

$$\begin{aligned} G_N(x) &= E_{\mathbf{y},\mathbf{D_N}}[L(y, h(x, \theta_{D_N}))] = E_{\mathbf{y},\mathbf{D_N}}[(y - \hat{y})^2] && \\ &= E_{\mathbf{y}}[(y - y^*)^2] + && \text{Noise} \\ &\quad (y^* - E_{\mathbf{D_N}}[\hat{y}])^2 + && \text{Bias} \\ &\quad E_{\mathbf{D_N}}[(\hat{y} - E_{\mathbf{D_N}}[\hat{y}])^2] && \text{Variance} \end{aligned} \tag{2.3}$$

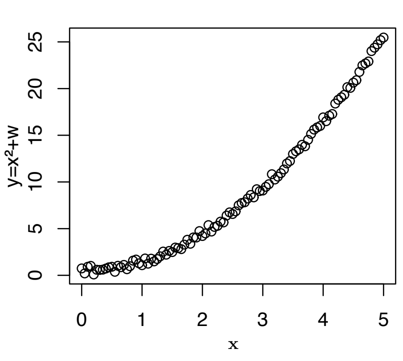
The first term $E_{\mathbf{y}}[(y - y^*)^2]$ only depends on the stochastic relationship $\mathbf{y}(x)$ between inputs and outputs, and characterizes the error coming from the approximation of $\mathbf{y}(x)$ at x. Ideally, one expects a deterministic relation between y and x. This ideal deterministic relation is, however, often "noisy" because of noise when getting empirical measures, or because of a lack of information that more precisely define the relation between x and y. This error component is therefore commonly referred to as *noise*, and is an irreducible error in a supervised learning process.

The second term $(y^* - E_{\mathbf{D_N}}[\hat{y}])^2$ quantifies the adequacy of the chosen model with respect to the optimal model. This term is called the "bias" of the model, sometimes referred to as "squared bias" as it is raised to the power two. In Example 2.3, the linear model is for instance biased as the true relationship is quadratic.

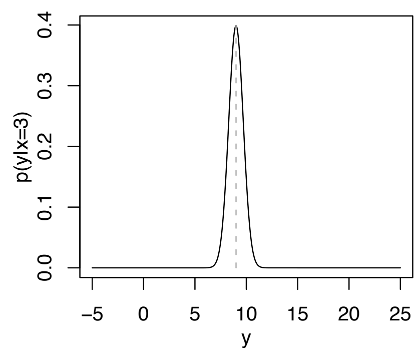
The third term $E_{\mathbf{D_N}}[(\hat{y} - E_{\mathbf{D_N}}[\hat{y}])^2]$ quantifies how much the predicted value of $\hat{y} = h(x, \theta_{D_N})$ will vary around the expected prediction value $E_{\mathbf{D_N}}[\hat{y}]$ of the model for different training set D_N. It therefore quantifies the sensitivity of the model prediction for a given training set. This term is referred to as the "variance" of the model.

Illustrative example

Precising Example 2.3, let us consider a stochastic process where the relation linking y to x is of the quadratic form $\mathbf{y}(x) = x^2 + \boldsymbol{\nu}$, where $\boldsymbol{\nu} \sim \mathcal{N}(0, 1)$ is a gaussian noise with mean 0 and standard deviation 1, and x takes values on the interval $[0; 5]$ (see figure 2.7(a)).



(a) Samples of the non-deterministic relation $y = x^2 + w$.



(b) The probability density $p(y|x{=}3)$.

Figure 2.7: Non-deterministic relation and uncertainty associated to a given x.

At one x, $\mathbf{y}$ follows a gaussian probability density with mean $y^* = E_{\boldsymbol{\nu}}[(x^2 + \boldsymbol{\nu})|x] = x^2$ and variance 1. This density is illustrated on figure 2.7(b), for $x = 3$.

Suppose that the model chosen to approximate this relation is a linear model of the form $\hat{y} = \theta_1 + \theta_2 x$, with parameters $\theta = (\theta_1, \theta_2)$, and that a training set D_N of $N = 100$ samples is available. A linear regression using the least square technique (see Section 2.2.3), allows to find the coefficients θ that minimize the quadratic loss function on a given training set D_N. Figure 2.8 illustrates four different realizations of this procedure for four different training sets D_N.

Depending on the training set D_N, the predicted value $\hat{\mathbf{y}} = h(x, \theta_{\mathbf{D_N}})$ at one point x is different. Figure 2.9 gives a graphical illustration of the noise, bias and variance quantities at one point x, where densities $p(y|x{=}3)$ and $p(\hat{y}|x{=}3)$ have been grouped on the same plot. The generalization error defined in Equation (2.3) equals the sum of the variance $\sigma_{\mathbf{y}}$ of $p(y|x)$ (the red curve), the variance $\sigma_{\hat{\mathbf{y}}}$ of $p(\hat{y}|x)$ (the blue curve), and the squared distance $d_{\mathbf{y}\hat{\mathbf{y}}}$ between the expected values of $p(y|x)$ and $p(\hat{y}|x)$ (the vertical dashed lines).

2.2.2 Learning procedure

Overview

The ideal outcome of the learning procedure is to produce the model with the lowest generalization error, i.e., the optimal model h^* as defined in Section 2.2.1. A number of practical issues usually prevent the procedure from finding this optimal model. In most cases, there is little or no a priori information about the analytical expression of the target operator, and the training set has a finite number of examples. The optimal model may have a high (possibly infinite) number of parameters, which cannot be estimated properly with a finite training set. At the same time, in the unlikely case where an infinite number of examples is available, computational issues would most probably prevent the processing of the learning algorithms. Simpler but biased models must therefore be considered in order to reduce the generalization error.

Assuming that the optimal model belongs to a space Λ^*, the common practice to handle the search for a model consists in decomposing the space of models Λ^* into a sequence

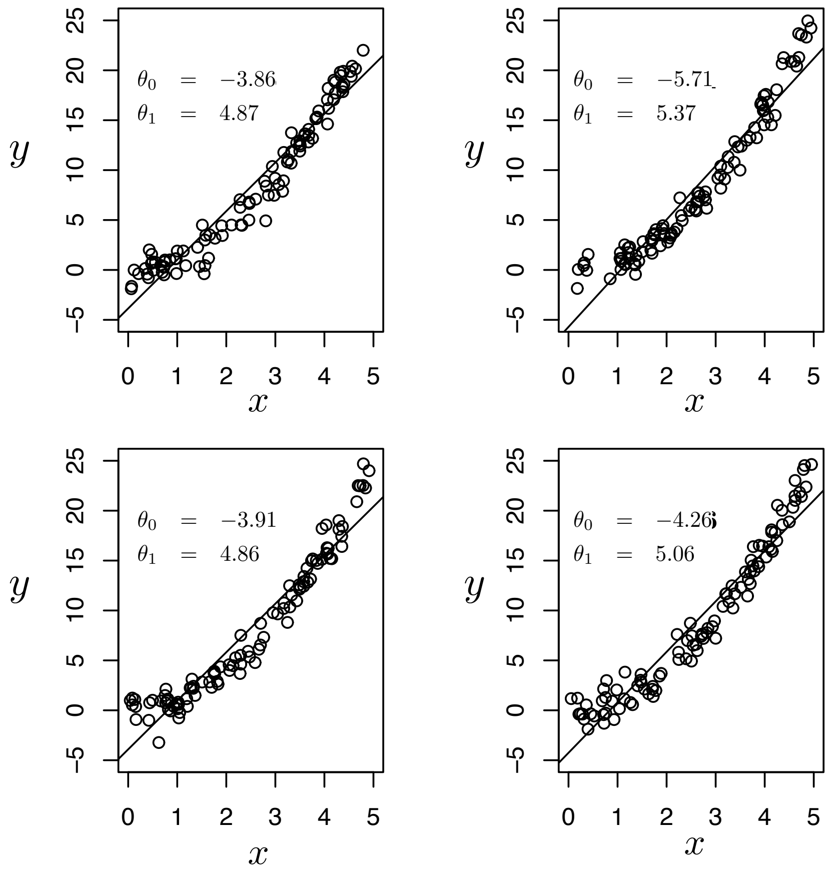


Figure 2.8: Four slightly different models due to variations in the training set.

of p nested spaces $\Lambda_1 \subset \Lambda_2 \ldots \subset \Lambda_p \subset \Lambda^*$ [8]. The search is then made at two nested levels, known as *parametric identification* and *structural identification* stages. The inner level consists in searching one of the subspace Λ_l, $1 \leq l \leq p$, and to produce the model that best fits the training data within this space. The task is usually formulated as an optimization problem whose objective function is to minimize the error made by the model on the training set. Techniques of parametric identification include for example the least square method for linear models, which we present in Section 2.2.3.

The outer level consists in assessing in terms of generalization error the performances of the model produced by the parametric identification stage for each subspace Λ_l. The assessment of the generalization is called the *validation*. Validation techniques include for example the cross validation, which we present in Section 2.2.2. The learning procedure eventually returns the model with the lowest estimated generalization error. Figure 2.10 illustrates the process.

Parametric identification

The parametric identification requires the definition of the following elements:

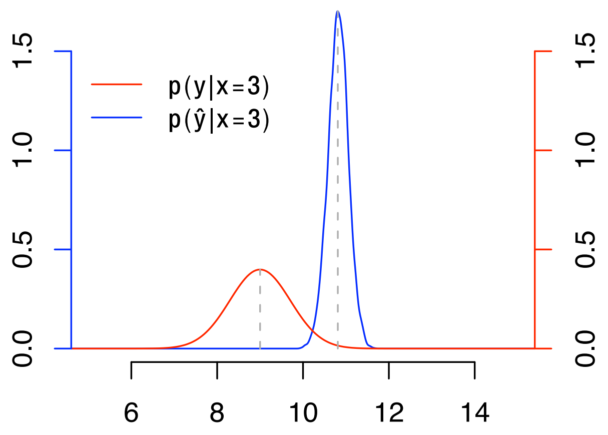


Figure 2.9: The probability densities $p(y|x{=}3)$ and $p(\hat{y}|x{=}3)$.

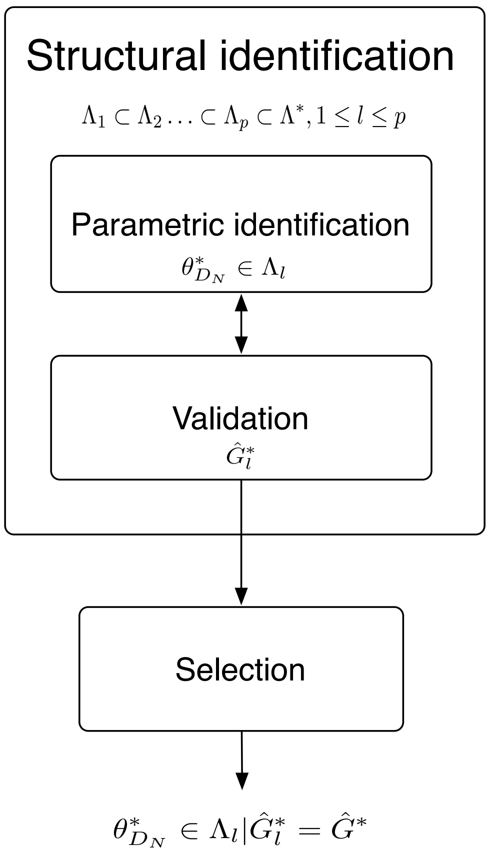

Structural identification
$\Lambda_1 \subset \Lambda_2 \ldots \subset \Lambda_p \subset \Lambda^*, 1 \le l \le p$
Parametric identification
$\theta^*_{D_N} \in \Lambda_l$
Validation
$\hat{G}^*_l$
Selection
$\theta^*_{D_N} \in \Lambda_l | \hat{G}^*_l = \hat{G}^*$

Figure 2.10: Decomposition of the learning procedure into its main components.

- A model space Λ_l
- A data set D_N of N observations.
- A loss function L. The loss of a model on a training set is called the *empirical error* E_{emp}, and is expressed as

$$E_{\text{emp}} = \frac{1}{N} \sum_{i=1}^{N} L(y_{[i]}, h_\theta(x_{[i]})).$$

- A learning algorithm $\mathcal{L}$, whose goal is stated as finding the model $h_\theta \in \Lambda_l$ that minimizes the empirical error.

Given these elements, the learning algorithm aims at finding the set of parameters θ that minimizes the empirical error. Denoting by $\theta^*_{D_N}$ the optimal parameters to emphasize the fact that they depend on the training set D_N, we have

$$\theta^*_{D_N} = \arg\min_\theta E_{\text{emp}} = \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} L(y_{[i]}, h(x_{[i]}, \theta)). \tag{2.4}$$

Depending on the class of models considered, the resolution of this optimization problem may be solved analytically, or by means of gradient descent techniques [68].

The use of the empirical error as the loss criterion in the objective function (Equation (2.4)) leads to identify models that fit the data rather than the underlying structure. It is therefore not a suitable estimate of the generalization error, as it does not account for the loss of the model for inputs not in the training set.

Structural identification

The goal of structural identification consists in

- Performing the parametric identification of a set of models with increasing complexity, by enlarging the model space.
- Assessing the generalization error by a validation procedure
- Selecting the model that has the lowest generalization error

The first stage consists in designing a nested sequence of model space $\Lambda_1 \subset \Lambda_2 \ldots \subset \Lambda_p \subset \Lambda^*$. A priori information on the nature of the target operator may be used to design this sequence. An example of such a sequence for a univariate model can be the set of polynomial models with $\hat{y} = \sum_{j=1}^{p} \theta_j x^j$ where each model space is defined by the parameter vector $\theta = (\theta_1, \theta_2, \ldots, \theta_p)$.

The validation procedure typically consists in partitioning the training set in two sets D_N^{tr} and D_N^{ts}. The first one is used for the parametric identification, and the second one, called test set, is used for assessing the loss of the model on examples that were not used for training the model. This way, it provides an estimate of the generalization error $\hat{G}_{ts}$ called test error.

A way to improve this estimate is to use K-cross validation, which consists in partitioning the training set D_N in K sets D_N^k of similar size. The validation procedure described above is repeated K times, using at each round one subset k as the test set $D_N^{ts} = D_N^k$, and the remaining $K-1$ training examples $D_N^{tr} = \cup_{k' \neq k} D_N^{k'}$ as the training set. This provides a set of 10 estimates $\hat{G}_k$, whose average $\hat{G}_{CV}$ is called the K-cross validation error.

Denoting by $\hat{G}^l$ the estimate of the generalization error for the model identified in the model space Λ_l, the structural identification loop generates a sequence of p estimates that typically have a parabolic form stemming from the bias-variance tradeoff.

Let $\hat{G}^*$ denote the lowest estimate of the generalization error. The model finally produced by the learning procedure is the model $h_\theta \in \Lambda_j$ from the space Λ_j for which $\hat{G}_l^* = \hat{G}^*$.

Data collection and preprocessing

We finish this discussion on the learning procedure by a few words on the data collection stage. The data collection stage is an important step of a learning task, and the quality of the data obtained at this stage strongly impacts on the ability of the learning process to properly identify the hidden relationships. Indeed, no matter the efficiency of the search procedure in the space of models, the learning process is bound to fail if the relationships are overly complex between input and output variables.

The data considered in this thesis will mainly be related to sensor measurements, such as temperature, humidity or pressure measurements. The quality of the data may be improved at the sensing stage by a careful calibration of the sensing devices.

Different kind of data preprocessing techniques may be further applied to improve the efficiency of the learning phase. First, techniques such as data filtering, detection and removal of erroneous data (also referred to as outliers), and inference of missing data may be used to improve the quality of the data. Second, feature extraction techniques may be used to transform the original set of variables in a set of variables more relevant for the learning task.

2.2.3 Classes of models

Linear models

Linear models have been widely used in statistics and remain today one of the most important tool [32]. Given an input $x \in \mathbb{R}^p$, the output of a linear model is predicted by:

$$\hat{y} = x^T \theta$$

where $\theta \in \mathbb{R}^p$ is the vector of parameters of the linear models. The most popular method to for the parametric identification of a linear model is the least square method, which is based on the L_2 quadratic loss function. Given a training set D_N, the minimization of the empirical gives the following objective function

$$\theta_{D_N} = \arg\min_\theta E_{\text{emp}}(\theta) = \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} L(y_{[i]}, h(x_{[i]}, \theta)).$$

Let X be the $N \times p$ matrix whose rows are the observations $x_{[i]}$, and Y the $N \times 1$ matrix whose elements are the outputs $y_{[i]}$, $1 \leq i \leq N$. Setting the derivative of $E_{\text{emp}}(\theta)$ to zero gives

$$\theta_{D_N} = (X^T X)^{-1} X^T Y$$

which is the standard solution to a regression problem.

Basis functions

Basis functions allow to rely on linear models to represent nonlinear relationships between inputs and outputs [32]. The core idea is to augment/replace the vector of inputs x with additional variables, which are transformations x, and then to use linear models in the new space formed with this transformed set of variables. Let us denote by

$$\begin{aligned} \pi_m : \mathcal{X} &\rightarrow \mathbb{R} \\ x &\mapsto x_m = \pi(x) \end{aligned}$$

the m-th transformation of the vector of input variables x. The transformation π_m is also called basis function, and the resulting model is a *linear basis expansion* in x. Some simple and widely used examples of π_m are the following:

- $\pi_m(x) = x_m$. This recovers the original model.
- $\pi_m(x) = x_{m'}^2$ or $\pi_m(x) = x_{m'}x_{m''}$ where $m', m'' \in \{1, \ldots, p\}$ are variables of the input vector x. This allows to augment the inputs with polynomial terms to achieve second-order Taylor expansion. Higher order may also be considered.
- $\pi_m(x) = \mathbf{1}(L_{m'} \leq x_{m'} \leq U_{m'})$, where $\mathbf{1}(.)$ is the indicator function. This allows to define regions over the variable $x_{m'}$, by breaking the range of $x_{m'}$ into multiple non overlapping areas.
- More generally, overlapping regions may also be designed. These are called kernel functions. A region is defined by a kernel $\pi_m(x)$, which is a non-negative function that maps inputs x to a non-negative number. A location can belong to several regions. Let l be the number of regions. The degree to which a location belongs to region m can be represented by the normalized kernel weight

$$\kappa_m(x) = \frac{\pi_m(x)}{\sum_{v=1}^{l} \pi_v(x)}$$

Basis expansions therefore provide a powerful way to extend the use of linear techniques to the modeling of nonlinear relationships. They however often require to add variables, and consequently parameters. Because of the bias variance tradeoff, the resulting performances of the model may decrease. Dimensionality reduction techniques (see Section 2.2.4) may be used concommitantly to address this issue.

Nonlinear models

Besides linear models with basis expansions, a number of other approaches have been designed over the years in the field of machine learning to model nonlinear relationships between data [32]. We briefly present the main nonlinear modeling approaches in the following.

Two general categories of approaches can be distinguished: global and local approaches. In global approaches, the learning procedure aims at fitting a single prediction model over the whole input space $\mathcal{X}$. Once the model is produced, it can be used to obtain predictions for any input $x \in \mathcal{X}$. Such approaches include neural networks [7] or support vector machines

[17] for example. A nice property of these approaches is that they provide a global representation of the relationship with one model, and have usually little storage requirements once the model is produced.

The fitting of a complex relationship is however often difficult in practice due to the bias/variance tradeoff. Local approaches address this issue by letting the learning procedure produce different models for different regions of the input space $\mathcal{X}$. The rationale of these approaches is to "divide and conquer", i.e., to partition the space in different regions in such a way that the relationship in each region may be represented by simpler models. Linear models are often used as the local models.

In local approaches, the partitioning of the space may be achieved by regression trees [10] or kernels [32] for example. Such a partitioning produces a collection of prediction models, one for each partition. The alternative to partitioning is the K-nearest neighbors approaches, where the learning procedure is postponed until a prediction for an input x is required. The region is then defined as the set of K closest neighbors of x in the training set. A local model is then built on the basis of the reduced training set. The size K of the neighborhood may be dynamically estimated on the basis of statistical properties, such as in the Lazy learning approach [8].

Nonlinear modeling techniques usually imply computationally intenseive learning procedures or large storage requirements. Their use in the context of sensor networks therefore conflict with the tight resources available on current sensor platforms. Therefore, although we mention the existence of these alternative modeling approaches for completing this section, these models shall not be specifically addressed again in this thesis.

2.2.4 Dimensionality reduction

Dimensionality reduction aims at reducing the number of inputs in supervised learning problems. As the number of parameters of a model is in most cases proportional to the number of inputs, reducing the dimensionality of a supervised learning problem is a way to trade variance for bias, with the aim of reducing the generalization error of a model.

Existing techniques for dimensionality selection include feature selection and feature transformation techniques. Feature selection approaches consist in selecting, out of the set of input variables, those that provide the greatest amount of information for predicting the output. For an up-to-date state of the art on feature selection see [31].

Feature transformation consists in transforming the input variables into a more compact set of variables, in such a way that important information for predicting the output is preserved. The resulting transformation can be mathematically formalized by means of basis functions.

We present in the following the most popular methods for feature selection and feature transformation, namely the best subset selection, forward selection and principal component analysis. Best subset selection and forward selection are instances of feature selection methods [31]. The Principal Component Analysis (PCA) is a classic dimensionality reduction technique in statistical data analysis, data compression, and image processing [42, 67]. The technique is based on a basis change which reduces the number of coordinates used to represent the data while maximizing the retained variance.

Algorithm 1 Best subset selection.

Input: Set of input variables $\mathcal{S} = \{x_1, x_2, \ldots, x_n\}$ and a learning procedure.
Output: Subset $\mathcal{S}^*$ that provides the lowest estimate of the generalization error.

```
1: for all $\mathcal{S}' \subset \mathcal{S}$ do
2:     Apply learning procedure with the subset of variables $\mathcal{S}'$
3:     Store $\hat{G}^*_{\mathcal{S}'}$, the lowest estimated error obtained.
4: end for
5: Return the subset $\mathcal{S}^* = \mathcal{S}'$ for which $\hat{G}^*_{\mathcal{S}'}$ is the lowest
```

Feature selection

Given a learning problem with a set $\mathcal{S} = \{x_1, x_2, \ldots, x_n\}$ of n input variables, the best subset approach consists in applying the learning procedure for all possible subsets of $\mathcal{S}$. The learning procedure is seen as a black box able to return (e.g. via cross-validation) an evaluation of the quality of a feature subset in terms of generalization error. The best subset is the one for which the estimate of the generalization error is the lowest. The algorithm is outlined in Algorithm 1.

When the number of variables is large, the best subset selection algorithm requires a high number of evaluations as 2^n subsets must be considered. For reducing the computational effort, an incremental approach called *forward selection* [32] is often used in practice. The algorithm, presented in Algorithm 2, has a maximum of n rounds, in which variables are incrementally added to a set of retained variables $\mathcal{S}^i_{kept}$. At each round, the added variable is the one in $\mathcal{S} \backslash \mathcal{S}_{kept}$ whose addition to the model provides the lowest estimate of the generalization error. The algorithm stops when the generalization error decreases due to overfitting. The maximum number of evaluations is reduced to $\frac{n(n+1)}{2}$. In practice, the method often performs as well as the best subset selection [32].

Principal component analysis

The PCA [42, 32] is a dimensionality reduction technique based on feature transformation. More specifically, the transformation is based on the projection of a multidimensional vector $x \in \mathbb{R}^n$ onto a subspace of dimension $q < n$. The subspace is defined by a set of orthonormal basis vectors $\{w_k\}$, $1 \leq k \leq q$, called *principal components* (PCs), which form the *PC basis*. The projections of x on the principal components are called the *principal component scores* $z \in \mathbb{R}^q$. Denoting by W the $[n \times q]$ matrix whose k-h column is w_k, the transform is expressed by

$$z = W^T x.$$

In PCA, the PCs are designed in order to minimize the approximation error entailed by the projections in the reduced space.

Given a set of N centered multivariate observations $x_{[i]} \in \mathbb{R}^n$ used as training samples[1],

[1]samples are centered so that the origin of the coordinate system coincides with the centroid of the set of samples. This translation is desirable to avoid a biased estimation of the basis $\{w_k\}_{1 \leq k \leq q}$ of $\mathbb{R}^n$ towards the centroid of the set of samples.

Algorithm 2 Forward selection.

Input: Set of input variables $\mathcal{S} = \{x_1, x_2, \ldots, x_n\}$ and a learning procedure.
Output: Subset $\mathcal{S}^*$ that provides the lowest estimate of the generalization error with the forward heuristic.

```
 1: $\mathcal{S}_{\text{remain}} = \mathcal{S}$
 2: $\mathcal{S}^0_{\text{keep}} = \emptyset$
 3: $\hat{G}^*_0 = \infty$
 4: $i = 0$
 5: **repeat**
 6:     **for all** $x_j \in \mathcal{S}_{\text{remain}}$ **do**
 7:         Apply the learning procedure with the subset of variables $\mathcal{S}^i_{keep} \cup x_j$
 8:         Store $\hat{G}^*_j$, the resulting estimate of the generalization error
 9:     **end for**
10:     $\hat{G}^*_{i+1} = \min_j \hat{G}^*_j$
11:     $x_{\text{select}} \leftarrow$ variable $x_j$ that gave the lowest $\hat{G}^*_j$
12:     $\mathcal{S}^{i+1}_{\text{keep}} \leftarrow \mathcal{S}^i_{\text{keep}} \cup x_{\text{select}}$
13:     $\mathcal{S}_{\text{remain}} \leftarrow \mathcal{S}_{\text{remain}} \backslash x_{\text{select}}$
14:     $i = i + 1$
15: **until** $G^*_i > G^*_{i-1}$
16: Return the subset $\mathcal{S}^* = \mathcal{S}^{i-1}_{\text{keep}}$
```

$1 \le i \le N$, the PCA aims at minimizing the following optimization function

$$\begin{aligned} J_q(w_k) &= \frac{1}{N}\sum_{i=1}^{N} ||x_{[i]} - \sum_{k=1}^{q} w_k w_k^T x_{[i]}||^2 \\ &= \frac{1}{N}\sum_{i=1}^{N} ||x_{[i]} - \hat{x}_{[i]}||^2 \end{aligned} \tag{2.5}$$

where the set of $q \le n$ vectors $\{w_k\}_{1 \le k \le q}$ of $\mathbb{R}^n$ have unit norm. The linear combinations $\hat{x}_{[i]} = \sum_{k=1}^{q} w_k w_k^T x_{[i]}$ represent the projections of $x_{[i]}$ on the PC basis, and Equation (2.5) therefore quantifies the mean squared error the original observations $x_{[i]}$ and their projections.

Given that the transform is linear, the PCA is particularly useful when data are correlated, as illustrated in Figure 2.11 where a set of $N = 50$ three-dimensional observations $x = (x_1, x_2, x_3) \in \mathbb{R}^3$ are plotted. Note that the correlation between the variables x_1 and x_2 is high, while the x_3 variables is independent of x_1 and x_2. The set of principal component (PC) basis vectors $\{w_1, w_2, w_3\}$, the two-dimensional subspace spanned by $\{w_1, w_2\}$ and the projections (crosses) of the original observations on this subspace are illustrated in the figure. We can observe that the original set of three-variate data can be well approximated by the two-variate projections in the PC space, thanks to the strong correlations between the variables x_1 and x_2.

The set of vectors $\{w_k\}_{1 \le k \le q}$ that minimizes (2.5) is obtained by setting to zero the derivative of $J_q(w_k)$ with respect to w_k. This gives [42]

$$\Sigma w_k = \lambda_k w_k \tag{2.6}$$

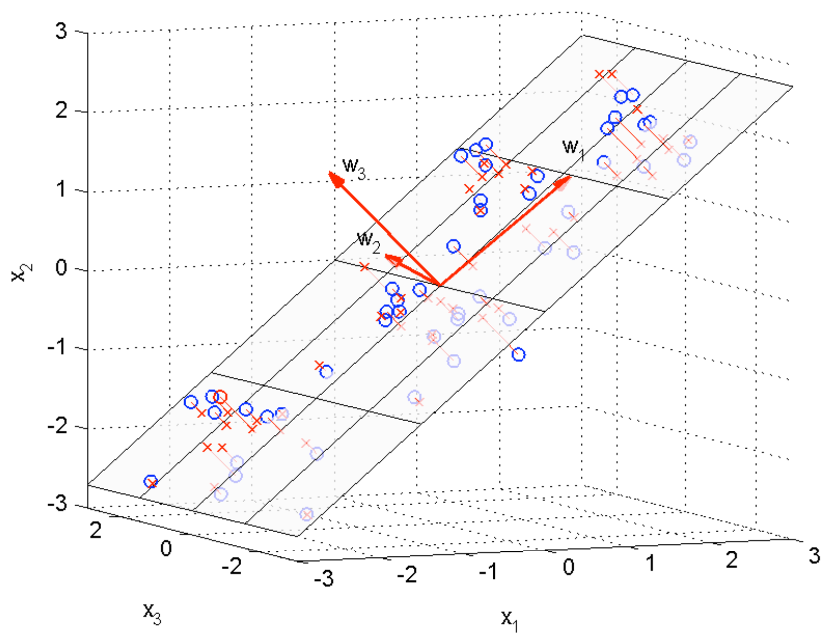


Figure 2.11: Illustration of the transformation obtained by principal component analysis. Circles denote the original observations while crosses denote their approximations obtained by projecting the original data on the two-dimensional subspace $\{w_1, w_2\}$ spanned by the two first principal components.

where $\Sigma = \frac{1}{N}\sum_{i=1}^{N} x_{[i]}x_{[i]}^T$ is the sample covariance matrix of the observations $x_{[i]}$. These vectors are therefore the first q eigenvectors $\{w_k\}$ of the sample covariance matrix Σ [42]. The corresponding eigenvalues λ_k quantify the amount of variance conserved by the eigenvectors. Indeed, left-multiplying Equation (2.6) by w_k^T gives

$$w_k^T \Sigma w_k = w_k^T \lambda_k w_k. \tag{2.7}$$

Since $w_k^T w_k = 1$, $w_k^T \Sigma w_k = w_k^T(\frac{1}{N}\sum_{i=1}^{N} x_{[i]}x_{[i]}^T)w_k = \sum_{i=1}^{N} \hat{x}_{[i]}^T \hat{x}_{[i]}$, and each eigenvalue λ_k quantifies the variance of the projections of the observations $x_{[i]}$ on the corresponding k-th eigenvector. The sum of the eigenvalues therefore equals the total variance of the original set of observations X, i.e.,

$$\sum_{k=1}^{n} \lambda_k = \frac{1}{N}\sum_{i=1}^{N} ||x_{[i]}||^2.$$

It is convenient to order the vectors w_k by decreasing order of the eigenvalues, so that the proportion P of retained variance by the first q principal components can be expressed by

$$P(q) = \frac{\sum_{k=1}^{q} \lambda_k}{\sum_{k=1}^{n} \lambda_k}. \tag{2.8}$$

Storing columnwise the set of vectors $\{w_k\}_{1 \le k \le q}$ in a $W_{n \times q}$ matrix, approximations $\hat{x}$ to x in $\mathbb{R}^n$ are obtained by

$$\hat{x} = WW^T x = Wz \tag{2.9}$$

where

$$z = W^T x = \begin{pmatrix} \sum_{j=1}^{n} w_{j1} x_j \\ ... \\ \sum_{j=1}^{n} w_{jq} x_j \end{pmatrix} = \sum_{j=1}^{n} \begin{pmatrix} w_{j1} x_j \\ ... \\ w_{jq} x_j \end{pmatrix} \tag{2.10}$$

denotes the column vector of coordinates of $\hat{x}$ in $\{w_k\}_{1 \leq k \leq q}$, also referred to as the *principal component scores.*

It is noteworthy that the set of vectors $\{w_k\}_{1 \leq k \leq q}$ which minimizes the mean squared error in Equation (2.5) is also the set of vectors which maximizes the retained variance by the approximations $\hat{x}_{[i]}$ [42, 67]. This results from the fact hat setting to zero the derivative of Equation (2.5) with respect to w_k comes down to maximizing

$$\begin{array}{rcl} J'_q(w_k) & = & \frac{1}{N} \sum_{i=1}^{N} || \sum_{k=1}^{q} w_k w_k^T x_{[i]} ||^2 \\ & = & \frac{1}{N} \sum_{i=1}^{N} ||\hat{x}_{[i]}||^2 \end{array}$$

which is the variance of $\hat{x}_{[i]}$. Therefore, the PCA provides the linear transform which not only minimizes the projection errors, but which also maximizes the retained variance.

2.2.5 Summary

This section covered the main supervised learning concepts and methods that will be used throughout this thesis. An emphasis was put on the importance of the bias/variance tradeoff. This tradeoff will be addressed at several occasions in the following chapters. In particular, most of the discussions concerning the benefits of modeling data in WSN will be driven by keeping this tradeoff in mind. This section also provided the reader with the basics of linear modeling, which will be encountered fairly often in the rest of this thesis, thanks to their low requirements in terms of computational and memory resources.

Chapter 3

Learning in Wireless Sensor Networks for Environmental Monitoring: State of the Art

The primary interest of learning techniques in the field of sensor networks is to reduce the amount of energy consumed by sensor nodes by reducing the amount of communication in the network. The radio is indeed the most energy consuming component on a sensor node, and sensor nodes are battery operated. Reducing the amount of communication is therefore one of the main factors for extending the lifetime of the network, which is an important requirement of environmental monitoring applications where sensor deployments are expected to run for months or years.

Learning techniques in sensor networks are attractive for the two following reasons. First, there usually exists a degree of redundancy in sensor network data. The redundancy stems from the fact that geographically close sensors are likely to collect similar measurements, and that measurements taken at two consecutive time instants are also likely to be similar. Learning allows to detect these redundancies, and to find mathematical models that represent the variations in sensor data in a more compact way. Second, it is rarely necessary to collect the exact measurements, and approximations can usually be tolerated by the observer. The lower the accuracy, the more compact the learning model, and therefore the higher the possible gains in communication. This makes possible the design of data collection strategies which trade accuracy for energy.

This chapter presents a review of the approaches that have been investigated for reducing the amount of communication in sensor networks by means of learning techniques. Section 3.1 gives the main definitions and notations, together with the metrics that will be used to assess the efficiency of learning techniques in improving the performances of environmental monitoring applications.

We classified the approaches based on learning in three groups, namely model-driven data acquisition, replicated models, and aggregative approaches. In model-driven approaches, the network is partitioned in two subsets, one of which is used to predict the measurements of the other one. The subset selection process is carried out at the base station, together with the computation of the models. Thanks to the centralization of the procedure, these approaches provide opportunities to produce both spatial and temporal models. Model-driven techniques can provide high energy savings as part of the network can remain in an idle mode. Their efficiency in terms of accuracy is however tightly dependent on the adequacy of the model to the sensor data. We present these approaches in Section 3.2.

Replicated models encompass a set of approaches where identical prediction models are run in the network and at the base station. The models are used at the base station to get the measurements of sensor nodes, and in the network to check that the predictions of the models are correct within some user defined ϵ. A key advantage of these techniques is to guarantee that the approximations provided by the models are within a strict error threshold ϵ of the true measurements. We review these techniques in Section 3.3.

Aggregation approaches allow to reduce the amount of communication by combining data within the network, and provide to a certain extent a mixture of the characteristics of model-driven and replicated models approaches. They rely on the ability of the network routing structure to aggregate information of interest on the fly, as the data is routed to the base station. As a result, the base station receives aggregated data that summarize in a compact way information about sensor measurements. The way data is aggregated depends on the model designed at the base station, and these approaches are therefore in this sense model-driven. The resulting aggregates may however be communicated to all sensors in the network, allowing them to check the approximations against their actual measurements, as in replicated models approaches. We discuss aggregative approaches in Section 3.4.

3.1 Problem statement

The main application considered in this thesis is the *periodic data collection*, in which sensors take measurements at regular time interval, and forward them to the base station [84]. Its main purpose is to provide the end-user, or observer, with periodic measurements of the whole sensor field. This task is particularly useful in environmental monitoring applications where the aim is to follow both over space and time the evolution of physical phenomena [84, 75].

Definitions and notations

The set of sensor nodes is denoted by $\mathcal{S} = \{1, 2, ..., S\}$, where S is the number of nodes. The location of sensor node i in space is represented by a vector of coordinates $c_i \in \mathbb{R}^d$, where d is the dimension of the space, typically 2 or 3. Besides the sensors and the battery, a typical sensor node is assumed to have a CPU of a few MHz, a memory of a few tens of kilobytes, and a radio with a throughput of a few hundreds of kilobits per second (cf. Section 2.1.1).

The entity of interest that is being sensed is called the *phenomenon*. The vector of the measurements taken in the sensor field at time t is denoted by $s[t] = (s_1[t], s_2[t], \ldots, s_S[t]) \in \mathbb{R}^S$. The variable $s[t]$ may be univariate or multivariate. Whenever confusion is possible, the notation $s_i[t]$ is used to specify the sensor node concerned, with $i \in \mathcal{S}$. The time domain is the set of natural numbers $\mathcal{T} = \mathbb{N}$, and the unit of time is called an *epoch*, whose length is the time interval between two measurements.

Most of the prediction models considered in this thesis will aim at approximating or predicting sensor measurements. Denoting by $\hat{s}_i[t]$ the approximation of $s_i[t]$ at time t, the prediction models will typically be of the form

$$\begin{array}{rcl} h_\theta : \mathcal{X} & \rightarrow & \mathbb{R} \\ x & \mapsto & \hat{s}_i[t] = h_\theta(x) \end{array}$$

The input x will in most cases be composed of the sensor measurements, sensor coordinates and/or time.

Example 3.1: *In many cases, there exists spatiotemporal dependencies between sensor measurements. A model may therefore be used to mathematically describe the relationship existing between one sensor and a set of other sensors. For example, a model*

$$\begin{aligned} h_\theta : \mathbb{R}^2 &\rightarrow \mathbb{R} \\ x = (s_j[t], s_k[t]) &\mapsto \hat{s}_i[t] = \theta_1 s_j[t] + \theta_2 s_k[t] \end{aligned}$$

can be used to approximate the measurement of sensor i on the basis of a linear combination of the measurements of sensors j and k, with $i, j, k \in \mathcal{S}$, and the parameters $\theta = (\theta_1, \theta_2) \in \mathbb{R}^2$. The use of such a model may allow to avoid collecting the measurement from sensor i if the model is assumed to be sufficiently accurate.

Example 3.2: *A model may aim at approximating the scalar field of the measurements. The input domain is the space and time domains $\mathbb{R}^d \times \mathbb{R}$, and the output domain is the set of real numbers, representing an approximation of the physical quantity at one point of space and time. For example, a linear combination of time and spatial coordinates $c_i \in \mathbb{R}^2$ in a 2D space domain can be used to create the following model*

$$\begin{aligned} h_\theta : (\mathbb{R}^2 \times \mathbb{R}) &\rightarrow \mathbb{R} \\ x = (c_i, t) &\mapsto \hat{s}_i[t] = x\theta^T \end{aligned}$$

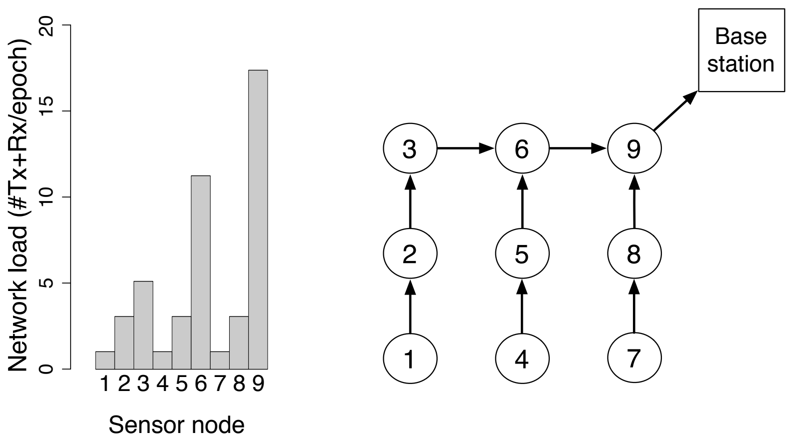
where $\theta = (\theta_1, \theta_2, \theta_3) \in \mathbb{R}^3$ are the weights of the linear combination which form the parameter vector θ. If the weights are known or can be estimated on the basis of past data, the model can be used to get approximated measurements of the sensors without actually collecting data from the network. The model can moreover be used to predict the measurements at future time instants, or at locations where sensors are not present.

We will assume that the data are retrieved from the network by means of a routing tree such as in Figure 3.1. The *observer* specifies the sampling frequency at which the measurements must be retrieved, using for example an aggregation service as described in Section 2.1.4.

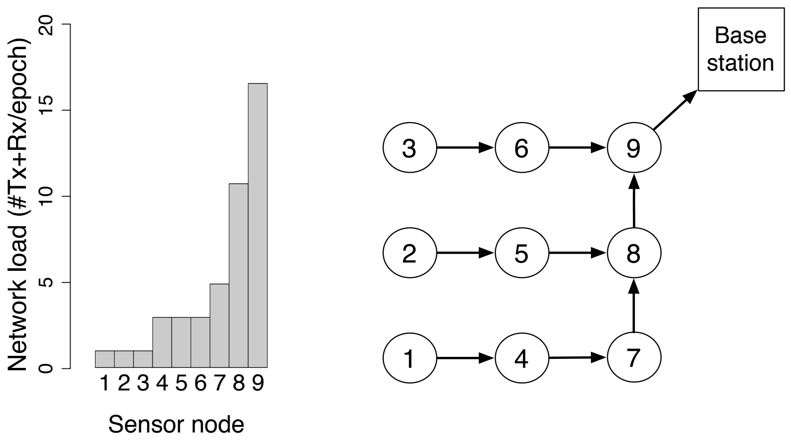
Network load and sensor lifetime

The purpose of prediction models is to trade data accuracy for communication costs. Prediction models are estimated by learning techniques, which use past observations to represent the relationships between measurements by means of parametric functions. The use of prediction models allows to provide the observer with approximations $\hat{s}[t]$ of the true set of measurements $s[t]$, and allows to reduce the amount of communication by either subsampling or aggregating data.

Given that the radio activity is the main source of energy consumption, the reduction of the use of the radio is the main way to extend the lifetime of a sensor network. In qualitative terms, **the lifetime is the time span from the deployment to the instant when the network can no longer perform the task** [82]. The lifetime is application specific. It can be for example, the instant when a certain fraction of sensors die, loss of coverage occurs (i.e., a certain portion of the desired area can no longer be monitored by any sensor) or loss of connectivity occurs (i.e., sensors can no longer communicate with the base station). For periodic data collection, the lifetime can be more specifically defined as *the number of data collection rounds until α percent of sensors die, where α is specified by the system designer* [76]. This definition makes the lifetime independent of the sampling rate at which data are collected from the network. Depending on α, the lifetime is therefore somewhere in between the number of rounds until the first sensor runs out of energy and the number of rounds until the last sensor runs out of energy.



(a) First configuration.



(b) Second configuration.

Figure 3.1: Network load sustained by each sensor node in a network of nine sensors, for two different routing trees. The bar plot (left) gives the total number of packets processed (receptions and transmissions) by each node in the network (right).

The communication costs related to the radio of a node i is quantified by **the network load L_i**, which is **the sum of the number of received and transmitted packets during an epoch**. Denoting by Rx_i and Tx_i the number of packet receptions and packet transmissions for node i during the epoch of interest, we have

$$L_i = \mathrm{Rx}_i + \mathrm{Tx}_i.$$

A network packet is assumed to contain one piece of information. A sensor transmitting its measurements and forwarding the measurement of another sensor therefore processes three packets during an epoch, i.e., one reception and two transmissions.

Figure 3.1 illustrates how the network load is distributed among sensors during an epoch. The network loads are reported for each sensor on the Left of Figure 3.1, for two different routing trees built such that the number of hops between the sensor nodes and the base station is minimized. Leaf nodes sustain the lowest load (only one transmission per epoch), whereas the highest load is sustained in both cases by the root node (8 receptions and 9 transmissions, totalizing a network load of 17 packets per epoch).

In data collection, it is important to remark that the nodes that are likely to have the lowest lifetime are the nodes close to the base station, as their radio activity is increased by the forwarding of data. The lifetime of these nodes is therefore closely related to the network lifetime, since once these nodes have run out of energy, the rest of network gets out of communication range of the base station. A particular attention will therefore be given in this thesis at the number of packets received and transmitted by the root node. More generally, we will often aim at quantifying the upper bound

$$L_{\max} = \max_i \ L_i \tag{3.1}$$

of the distribution of the network loads in the network. Most of the methods and techniques discussed in this thesis will aim at reducing this quantity, which will be referred to as **highest network load**. In order to consider the effects of collisions, interference or radio malfunctioning, we will use orders of magnitudes instead of precise counting of the packets. Without the use of learning strategies, the order of magnitude of the highest network load is in

$$L_{\max} \sim O(S)$$

where S is the number of nodes.

Data accuracy

The quantification of the error implied by a prediction model is an important practical issue, as in many cases the observer needs to know how far the predictions obtained at the base station are from the true measurements. Three levels of accuracy will be encountered in the thesis.

First, *probabilistic bounded approximation errors* will refer to approximations where

$$P(|s_i[t] - \hat{s}_i[t]| > \epsilon) = 1 - \delta, \ \forall i \in \mathcal{S}, t \in \mathcal{T} \tag{3.2}$$

which guarantees that, with probability $1 - \delta$, approximations do not differ by more than ϵ from the true measurements. The observer can set the error threshold ϵ and the probability guarantee δ.

Second, *bounded approximation errors* will refer to approximations where

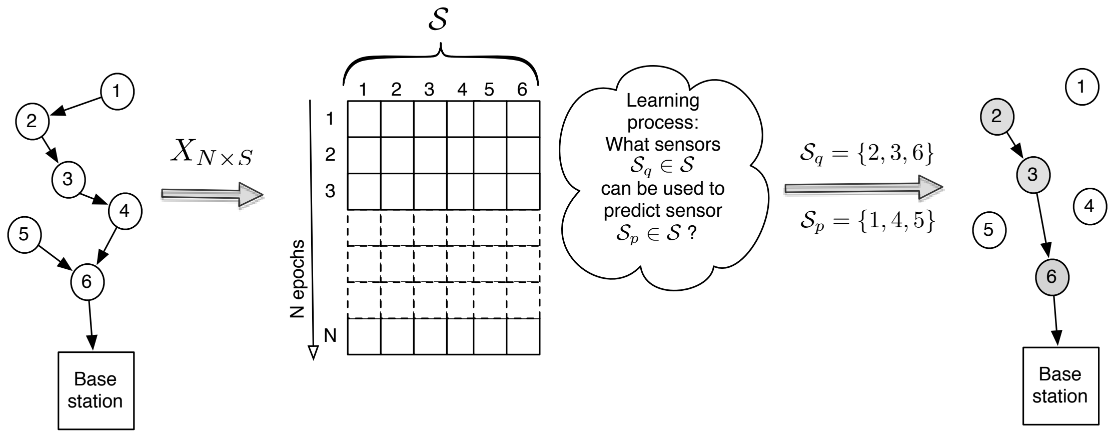
$$|s_i[t] - \hat{s}_i[t]| < \epsilon, \ \forall i \in \mathcal{S}, t \in \mathcal{T} \tag{3.3}$$

which ensures the observer that all approximations $\hat{s}_i$ obtained at the base station are within $\pm\epsilon$ of the true measurement $s_i[t]$. This level of accuracy is the highest, as it allows the observer to precisely define the tolerated error threshold.

Finally, *unbounded errors* will refer to modeling schemes where there is no bound between the approximations $\hat{s}_i[t]$ obtained at the base station and the true measurement $s_i[t]$ taken by sensor i.

3.2 Model-driven acquisition

In model-driven data acquisition [19, 20], a model of the sensor measurements is estimated at the base station, and used to optimize the acquisition of sensor readings. The rationale of the approach is to acquire data from sensors only if the model is not sufficiently rich to provide the observer with the requested information. An overview of the approach is presented in Figure 3.2. In a first stage, measurements are collected over N epochs from the whole network at the base station, and stored in a matrix X of dimension $N \times S$, where column j contains the measurements from sensor j over the N epochs, and row i contains the measurements from the whole network during the i-th epoch. The data set X is then used to estimate a model able to answer the users' queries without collecting data from the whole network. More precisely, the model-driven system aims at finding a subset of sensors $\mathcal{S}_q \subseteq \mathcal{S}$ from which the measurements of the other sensors $\mathcal{S}_p = \mathcal{S} \setminus \mathcal{S}_q$ can be predicted. The subscript p and q refer to the *queried* and *predicted* subsets. Once the subsets $\mathcal{S}_q$ and $\mathcal{S}_p$ have been identified, measurements are only collected from the sensors in $\mathcal{S}_q$.



$\mathcal{S}$
1 2 3 4 5 6
1
2
3
N
N epochs
$X_{N\times S}$
Learning process: What sensors $\mathcal{S}_q \in \mathcal{S}$ can be used to predict sensor $\mathcal{S}_p \in \mathcal{S}$?
$\mathcal{S}_q = \{2,3,6\}$
$\mathcal{S}_p = \{1,4,5\}$
Base station
Base station

Figure 3.2: Model-driven approach: the learning process takes place at the base station. It aims at finding a subset of sensors from which the measurements of the other sensors can be predicted.

In practice: Model-driven approaches are particularly appropriate for scenarios where groups of sensor nodes have correlated measurements. One node of the group sends its measurements to the base station, which uses them to predict the measurements of other nodes in the group. Thanks to the fact that the base station has a global view of the measurements collected, model-driven approaches allow to detect correlation between sensor nodes that may be far away in the network. A priori information on the periodicity or stationarity of the measurements can be used to determine how many observations N should be collected before using the model. For example, a network collecting outdoor temperature measurements is likely to exhibit diurnal patterns. If the patterns are consistent over days, observations can be taken over a one day period and used to model the measurements of the following days.

Optimization problem

The model used at the base station aims at predicting a vector of measurements $\hat{s}_p[t]$ for sensors in $\mathcal{S}_p$ with a prediction model

$$\begin{aligned} h_\theta : \mathbb{R}^{|\mathcal{S}_q|} &\rightarrow \mathbb{R}^{|\mathcal{S}_p|} \\ s_q[t] &\mapsto \hat{s}_p[t] \end{aligned} \tag{3.4}$$

where the input $\hat{s}_q[t]$ is the vector of measurements collected from sensors in $\mathcal{S}_q$ at time t, and θ is a vector of parameters. The model-driven approach allows to trade energy for accuracy by carefully choosing the subsets $|\mathcal{S}_q|$ and $|\mathcal{S}_p|$.

Costs: The cost associated to the query of a subset of sensors $\mathcal{S}_q$ is denoted $C(\mathcal{S}_q)$, and aims at quantifying the energy required to collect the measurements from $C(\mathcal{S}_q)$. The cost is divided into *acquisition* and *transmission* costs in [19, 20]. Acquisition refers to the energy required to collect a measurement, and transmission to the energy required for sending the measurement to the base station. The transmission costs are in practice difficult to estimate, because of multi-hop routing and packet loss issues. A simple and qualitative metric is to define the cost as the number of sensors in $\mathcal{S}_q$.

Accuracy: Let $s_{p_i}[t]$ and $\hat{s}_{p_i}[t]$ be the true measurement and the prediction for the i-th sensor in $\mathcal{S}_p$ at time t. The accuracy associated to $\hat{s}_{p_i}$ is denoted $R(\mathcal{S}_{p_i})$, and the accuracy associated to the vector of prediction $\hat{s}_p$ is denoted $R(\mathcal{S}_p)$. Different choices are possible to define how accuracy is quantified. In [19, 20], authors suggest using

$$R(\mathcal{S}_{p_i}) = P(s_{p_i}[t] \in [\hat{s}_{p_i}[t] - \epsilon, \hat{s}_{p_i}[t] + \epsilon]) \tag{3.5}$$

where $i \in \mathcal{S}_p$ and ϵ is a user-defined error threshold. This accuracy metric quantifies the probability that the true measurement $s_{p_i}[t]$ is within $\pm\epsilon$ of the prediction $s_{p_i}[t]$. The overall accuracy of the vector of prediction $\hat{s}_p[t]$ is defined as the minimum of $\hat{s}_{p_i}[t]$, i.e.,

$$R(\mathcal{S}_p) = \min_i R(\mathcal{S}_{p_i}). \tag{3.6}$$

Optimization loop: The goal of the optimization problem is to find the subset $\mathcal{S}_q$ that

1. minimizes $C(\mathcal{S}_q)$,
2. such that $R(\mathcal{S}_p) > 1 - \delta$

where δ is a user-defined confidence level. An exhaustive search among the set of partitions $\{\mathcal{S}_q, \mathcal{S}_p\}$ can be computationally expensive. There exists 2^S combinations for the set of predicted sensors, and S can be large. In order to speed up this process, an incremental search procedure similar to the forward selection algorithm (Section 2.2.4) can be used. The search is initialized with an empty set of sensors $\mathcal{S}_q = \emptyset$. At each iteration, for each $i \in \mathcal{S}_p$, the costs $C(\mathcal{S}_q \cup i)$ and accuracy $R(\mathcal{S}_p \setminus i)$ are computed. If one sensor i can be found such that $R(\mathcal{S}_p \setminus i) > 1 - \delta$, the procedure returns $\mathcal{S}_q \cup i$ as the set of sensors to query and $\mathcal{S}_p \setminus i$ as the set of sensors to predict. Otherwise, the sensor node that provided the *best* tradeoff is added to the subset $\mathcal{S}_q$ and removed from $\mathcal{S}_p$. The *best* sensor node is the node that maximizes the ratio $\frac{R(\mathcal{S}_p \setminus i)}{C(\mathcal{S}_q \cup i)}$ [19, 20].

Multivariate Gaussians

Different learning procedure can be used to compute the model h_θ in (3.4). Authors in [19, 20] suggest the use of a multivariate Gaussian to represent the set of measurements, which allows to compute predictions and confidence bounds using computationally efficient matrix products. Denoting by $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_S) \in \mathbb{R}^S$ the random vector of the measurements, where the i-th value represents the measurement of the i-th sensor, the Gaussian probability density function (pdf) of $\mathbf{s}$ is expressed as (cf. Appendix A)

$$p(\mathbf{s} = s) = \frac{1}{\sqrt{(2\pi)^S |\Sigma|}} \exp^{\left(-\frac{1}{2}(s-\mu)^T \Sigma^{-1} (s-\mu)\right)}$$

where μ and Σ are the mean and covariance matrix of the random vector $\mathbf{s}$ (Figure 3.2). Figure 3.3 illustrates a Gaussian over two correlated variables $\mathbf{s}_1$ and $\mathbf{s}_2$. For a Gaussian, the mean is the point at the center of the distribution, and the covariance matrix Σ characterizes the spread of the distribution. More precisely, the i-th element along the diagonal of Σ, $\sigma_{(i,i)}$, is the variance of the i-th variable, and off-diagonal elements $\sigma_{(i,j)}$ characterize the covariances between the pairs (i, j) of variables. A high covariance between two variables means that their measurements are correlated, such as variables $\mathbf{s}_1$ and $\mathbf{s}_2$ in Figure 3.3.

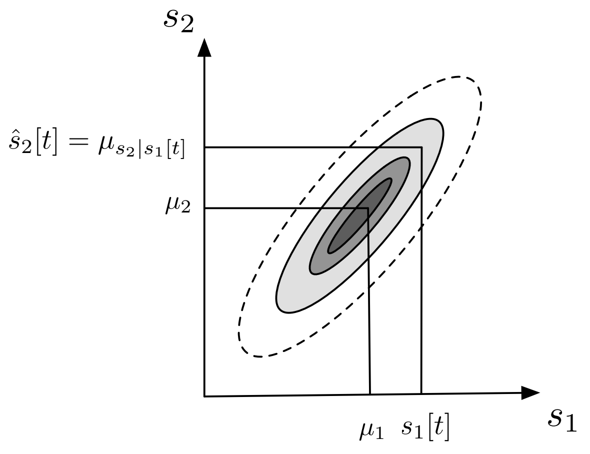


Figure 3.3: Gaussian model of two correlated variables. Knowledge of the outcome of $\mathbf{s}_1$ allows to better estimate the outcome of $\mathbf{s}_2$ thanks to conditioning.

When sensor measurements are correlated, information on some measurements constrains the values of other measurements to narrow probability bands. Let $s_q[t] \in \mathcal{S}_q$ and $s_p[t] \in \mathcal{S}_p$ be the vectors of measurements of sensors in $\mathcal{S}_q$ and $\mathcal{S}_p$ at time t, with $\mathcal{S}_p = \mathcal{S} \setminus \mathcal{S}_q$. The Gaussian pdf

$$p(s_p|s_q[t]) = \frac{1}{\sqrt{(2\pi)^{|\mathcal{S}_p|} |\Sigma|}} \exp^{\left(-\frac{1}{2}(s_p-\mu_{p|q})^T \Sigma_{p|q}^{-1} (s_p-\mu_{p|q})\right)}$$

of the random variable $\mathbf{s}_p$ can be computed using the mean μ and covariance matrix Σ of the model $p(s)$. This computation, called conditioning, gives [68]

$$\begin{aligned}\mu_{p|q} &= \mu_p + \Sigma_{pq}\Sigma_{qq}^{-1}(s_q[t] - \mu_q)\\ \Sigma_{p|q} &= \Sigma_{pp} - \Sigma_{pq}\Sigma_{qq}\Sigma_{qp}.\end{aligned}$$

where Σ_{pq} denotes the matrix formed by selecting the rows $\mathcal{S}_p$ and the columns $\mathcal{S}_q$ from the matrix Σ. After conditioning, the best approximations $\hat{s}_p[t]$ to sensors in $\mathcal{S}_p$ are given by the mean vector

$$\hat{s}_p[t] = \mu_{p|q} \tag{3.7}$$

The probability

$$P\left(s_i[t] \in [\hat{s}_i[t] - \epsilon, \hat{s}_i[t] + \epsilon]\right) \tag{3.8}$$

depends on the variance of the measurements of the sensors in $\mathcal{S}_p$ after the conditioning. These variances are actually known as they are the diagonal elements of the covariance matrix $\Sigma_{p|q}$, and allow to estimate the quantity (3.8) by referring to a student's t table [53].

Discussion

With model driven data acquisition, the communication savings are obtained by reducing the number of sensors involved in the data collection task. These approaches can provide important communication and energy savings, by leaving the set of sensors in $\mathcal{S}_p$ in an idle mode. For continuous queries however, the constant solicitation of the same subset of sensors leads to an unequal energy consumption among nodes. This issue can be addressed by recomputing from time to time the subset of sensors $\mathcal{S}_p$ in such a way that sensors whose remaining energy is high are favored.

In terms of communication savings, the network load is reduced by a factor that depends on the ratio of the total number of sensors over the number of queried sensors. The highest load is in

$$L_{\max} \sim O(|\mathcal{S}_q|)$$

and is sustained by the root node of the tree connecting the sensors in $\mathcal{S}_q$. The main issue of the model-driven approach probably lies in the assumption that the model is correct, which is a necessary condition for the scheme to be of interest in practice. The model parameters μ and Σ must be estimated from data, and any changes in the data distribution will lead to unbounded errors. Also, the presence of outliers in the queried measurements may potentially strongly affect the quality of the predictions. Model-driven data acquisition therefore allows potentially high communication savings, but is little robust to unexpected measurement variations and non-stationary signals.

3.3 Replicated models

Replicated models (RM) form a large class of approaches allowing sensor nodes to remove temporal or spatial redundancies between sensor measurements in order to decrease the network load. They were introduced in the field of sensor networks by Olston in 2001 [72]. Their rationale consists in having identical prediction models running both within the network and at the base station. Other names have been given to this approach in the literature, such as *approximate caching* [72], *dual Kalman filters* [40], *replicated dynamic* *probabilistic models* [15], *dual prediction scheme* [79] or *model-aided approach* [94]. We will adopt here the *replicated models* denomination since it is the one which better expresses in our sense the common rationale of these approaches.

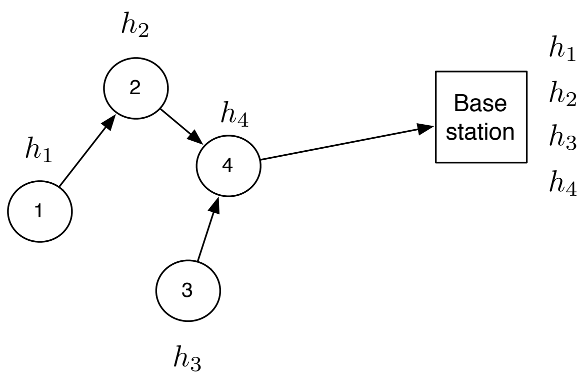


Figure 3.4: Replicated models: Only the models are communicated to the base station. As long as the model is correct, no communication is necessary.

Figure 3.4 gives an illustration of the replicated model approach. Four models h_i are used to predict the sensor measurements, one for each sensor i. The base station and the sensors use the same models. At the base station, the model is used to predict the measurements. On the sensor nodes, the model is used to compute the same prediction as the base station, and to compare it with the true measurement. If the prediction is more than a user-defined ϵ away from the true measurement, a new model is communicated. RM approaches therefore guarantee the observer with bounded approximation errors. Replicated models may be used to predict measurement both over the temporal and spatial domains. We first cover the approaches where models only take into account temporal variations, and then present the strategies that have been investigated to extend RM to the modeling of spatial variations.

In practice: In many environmental monitoring applications, the observer can tolerate approximations for the collected measurements. For example, in plant growth studies, ecologists reported that it is sufficient to have accuracy of $\pm 0.5°C$ and 2% for temperature and humidity measurements, respectively [20]. Replicated models provide an appropriate mechanism to deal with these scenarios.

Temporal modeling

Temporal modeling with RM is the simplest approach, as sensors independently produce prediction models for their own measurements [72, 40, 79]. Models are of the kind

$$\hat{s}_i[t] = h_i(x, \theta)$$

where x is a vector of inputs that consists, for example, in the past measurements of sensor i, and θ is a vector of parameters. There is one model for each sensor node i, whose task is to produce predictions for the upcoming measurements of the sensor node. Once such a model is produced, it is communicated to the base station, which uses it to infer the actual measurements taken by sensor i.

Sensor nodes use their own copy of the model to compare the model predictions with the sensor measurements. The model is assumed to be correct as long as the predicted and the true measurements do not differ by more than a user defined error threshold ϵ, i.e.,

$$|s_i[t] - \hat{s}_i[t]| < \epsilon.$$

Once the prediction is more than ϵ away from the true measurement, an update is sent to the base station to notify that the error threshold is not satisfied. The update may consist of the measurement, or of the parameters of a new model built on the basis of the more recent measurements. This way, replicated models guarantee the observer that all measurements provided at the base station within $\pm\epsilon$ of the true measurements.

Replicated models allow to reduce communication since, as long as the model predictions are within $\pm\epsilon$ of the true measurement, no updates occur between the sensor nodes and the base station. The inputs of the model, if they depend on the sensor measurements, are inferred by the base station using the measurements predicted by the model. The sensor nodes also use past predicted measurements as inputs to their models. This way, the sensor nodes and the base station apply exactly the same procedure. Sensor measurements are sent only when a model update is needed.

Algorithm 3 RM - Replicated model algorithm.

Input:
ϵ: Error threshold.
h: Model.

Output:
Packets containing model updates sent to the base station (BS).

```
1: $t \leftarrow 1$
2: $\theta[t] \leftarrow$ init(h)
3: $\theta^{\text{last}} \leftarrow \theta[t]$
4: sendNewModel($h, \theta^{\text{last}}$) to BS
5: while True do
6:     $t \leftarrow t + 1$
7:     $s[t] \leftarrow$ getNewReading()
8:     $\hat{s}[t] \leftarrow$ getPrediction($h, \theta^{\text{last}}$)
9:     $\theta[t] \leftarrow$ update(h, $\theta$[t − 1], s[t])
10:    if $|\hat{s}[t] - s[t]| > \epsilon$ then
11:        $\theta^{\text{last}} \leftarrow \theta[t]$
12:        sendNewModel($h, \theta^{\text{last}}$) to BS
13:    end if
14: end while
```

The pseudocode for running replicated models on a sensor node is given by Algorithm 3. The subscript i is dropped for the sake of clarity. On the base station, the scheme simply consists in using the most recently received model to infer the sensor's measurements. Four types of techniques have been proposed to run temporal RM [72, 47, 41, 87, 79], that are presented in detail in the following. The following summary gives an overview of their differences:

- Constant model [72, 47]: The most simple model, nonetheless often efficient. It does not rely on any learning procedure, and as result cannot represent complex variations.

- Kalman filter [41]: The technique provides a way to filter the noise in the measurements.
- Autoregressive model [87]: It allows to predict more complex variations than the constant model, but requires a learning stage.
- Least mean square filter [79]: The filter is based on an autoregressive model which adapts its coefficients over time.

Constant model: In [72, 47], RM are implemented with constant prediction models

$$\hat{s}_i[t] = s_i[t-1].$$

Although simple, the constant model is well suited for slowly varying time series. Also, it has the advantage of not depending on any parameters. This keeps the size of an update to a minimum, which consists only in the new measurement $s_i[t]$. This makes the constant model bound to reduce the communication between the sensor and the base station. Figure 3.5 illustrates how a constant model represents a temperature time series. The time series was obtained from the Solbosch Greenhouse on the 16th of August, 2006. Data were taken every 5 minutes, for a one day period, giving a set of 288 measurements. The measurements are reported with the red dashed lines, and the approximations obtained by a constant model with an error threshold of $\epsilon = 1°C$ are reported with the black solid line. Updates are marked with black dots at the bottom of the figure. Using RM the constant model allows to reduce to 43 the number of measurements transmitted, resulting in about 85% of communication savings.

Kalman filter: In [41], authors suggested to use Kalman filters as a generic approach to modeling sensor measurements for replicated models. A Kalman filter is a stochastic, recursive data filtering algorithm introduced in 1960 by R.E. Kalman [43]. It can be used to estimate the dynamic state of a system at a given time t by using noisy measurements issued at time $t_0 : t-1$.

The main goal of a Kalman filter is to provide good estimations of the internal state of a system by using a priori information on the dynamic of the system and on the noise affecting the measurements. The system model is represented in the form of the following equations

$$\mathbf{x}[t] = F\mathbf{x}[t-1] + \boldsymbol{\nu}_p[t] \tag{3.9}$$

$$\mathbf{s}[t] = H\mathbf{x}[t-1] + \boldsymbol{\nu}_m[t] \tag{3.10}$$

where $\mathbf{x}[t]$ is the internal state of the process monitored, which is unknown, and $\mathbf{s}[t]$ is the measurement obtained by a sensor at time instant t. The matrix F is the state transition matrix, which relates the system states between two consecutive time instants. The matrix H relates the system state to the observed measurement. Finally, $\boldsymbol{\nu}_p[t]$ and $\boldsymbol{\nu}_m[t]$ are the process noise and measurement noise, respectively.

The state of the system is estimated in two stages. First, a *prediction/estimation* stage is used to propagate the internal state of the system by means of Equation (3.10). Second, a *correction stage* fine-tunes the prediction step by incorporating the actual measurement, in

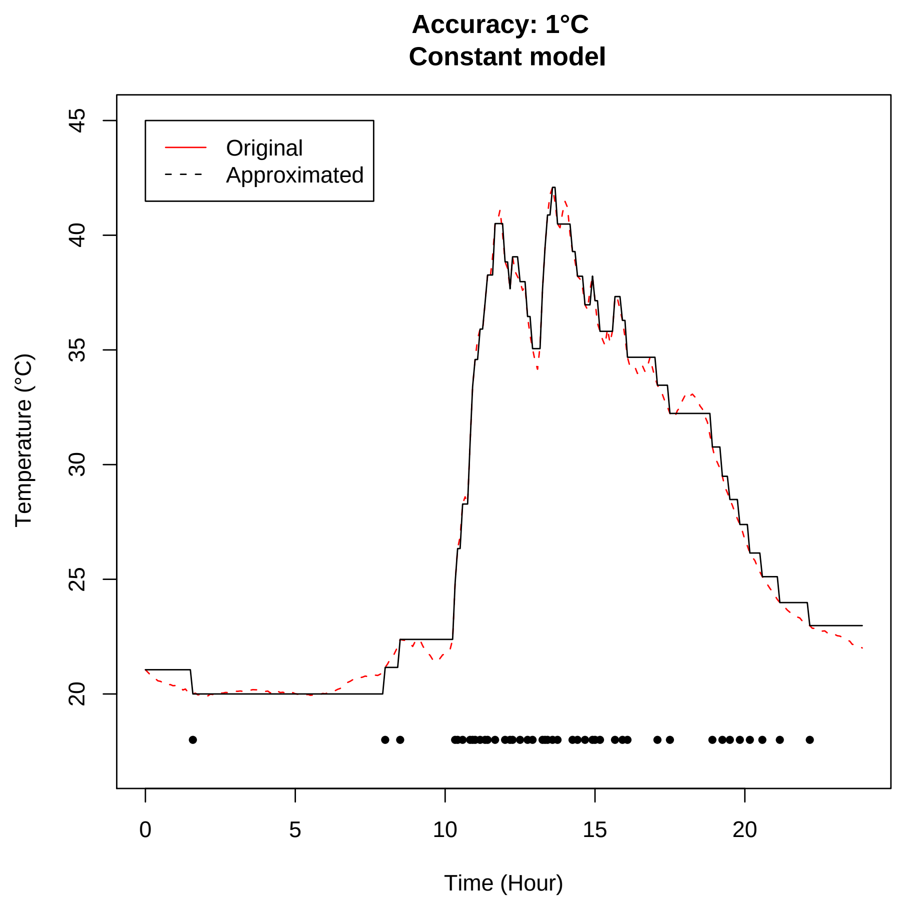


Figure 3.5: A constant model acting on a temperature time series from the Solbosch greenhouse (16th of August 2006), with a constant model and an error threshold set to $\epsilon = 1°C$

such a way that the error covariance matrix between the measurements and the predictions is minimized. Eventually, a prediction $\hat{s}[t]$ is obtained, expected to be closer to the true state of the system than the actual measurement $s[t]$. We do not go in the details of the mathematical steps (which can be found in [41]), as the approach raises the main following issue.

The purpose of Kalman Filters is to filter the noise in a signal, in order to get the best possible estimate, in a mean square sense, of the state of the underlying process of the measurements. The goal of replicated models is however not stated in terms of noise reduction, but in terms of ϵ approximation to the measurements. Therefore, KFs do not quite fit the problem considered.

Autoregressive models: In [87], authors suggest the use of autoregressive models, a well known family of models in time series prediction [14, 11]. An autoregressive model is a function of the form

$$\hat{s}[t] = \theta_1 s[t-1] + \theta_2 s[t-2] + \ldots + \theta_p s[t-p] \tag{3.11}$$

that aims at predicting the measurement at time t by means of a linear combination of the measurements collected at the previous p time instants. The vector of parameters has p elements $\theta = (\theta_1, \theta_2, \ldots, \theta_p)^T$, and p is called the order of the model. Using the notations $x[t] = (s[t-1], s[t-2], \ldots, s[t-p])^T$ to denote the vector of inputs of the model at time instant k, the relationship can be written as

$$\hat{s}[t] = x[t]^T \theta. \tag{3.12}$$

The estimation of the vector of parameters θ is obtained by first collecting N measurements, and by applying the standard procedure of regression θ (cf. Section 2.2.3) on the sensor node. The set of parameters is then communicated to the sink, and used until the prediction error becomes higher than the user predefined threshold ϵ. A new model is then sent to the base station. Authors in [87] also considered the fact that a measurement not correctly predicted by the model may be an outlier, and suggested to use statistical tests to determine wether or not to send an update.

Least Mean Square filter: In [79], authors argue that the adaptive filter theory [4] offers an alternative solution for performing predictions, without requiring a priori information about the statistical properties of the phenomenon of interest. Among the variety of techniques, they chose the Least Mean Square (LMS) algorithm, arguing that it is known to provide good performances in a wide spectrum of applications. As in Equation (3.12), the LMS is an autoregressive filter, where the output is a linear combination of previous measurements $\hat{s}[t] = x[t]^T \theta[t]$. In contrast to the proposed approach by [87], the vector of parameters $\theta[t]$ is updated over time as new measurements are taken. The LMS theory gives the following set of three equations for updating the parameters:

$$\begin{aligned} \hat{s}[t] &= x[t]^T \theta[t] \\ e[t] &= s[t] - \hat{s}[t] \\ \theta[t+1] &= \theta[t] + \mu x[t]^T e[t]. \end{aligned}$$

where $e[t]$ is the error made by the filter at epoch t, and μ is a parameter called the step size, which regulates the speed of convergence of the filter. The choice of the order p of the filter and of the step size μ are the only parameters that must be defined a priori. Authors of [79] suggested on the basis of their experiments that orders from 4 to 10 provided good results. Concerning the choice for the parameter μ, they suggest to estimate it by setting $\mu = 10^{-2} \frac{1}{E}$ where $E = \frac{1}{T} \sum_{t=1}^{T} |s[t]|^2$ is the mean input power of the signal.

Spatial modeling

Two different approaches were investigated to model spatial dependencies with replicated models. In [80], replicated models are used on each edge of a routing tree that connects the nodes to the base station (edge monitoring). In [15], the network is partitioned in groups of nodes, with one model for each group (clique models).

Edge monitoring: In this approach [80], it is assumed that nodes are connected to the base station by means of a routing tree. As with temporal modeling, there is one model h_i for each sensor i. Additionally, there is also one model for each edge connecting neighboring nodes in the routing tree. More specifically, assuming that there is an edge connecting node

i to j, a model h_{j-i} is also produced by node j to represent the difference in value between the measurement $s_j[t]$ and the approximation $\hat{s}_i[t]$. Denoting by $\Delta_{j-i}[t] = s_j[t] - \hat{s}_i[t]$ this difference, the model has the form

$$\hat{\Delta}_{j-i}[t] = h_{j-i}(x, \theta)$$

where x is a vector of inputs that consists for example in the past difference between measurements of sensor i and j, and θ is a vector of parameters. The most simple model is the constant model, which was the only one considered in [80], and which is defined by

$$\hat{\Delta}_{j-i}[t] = \Delta_{j-i}[t-1].$$

Note that more complex models such as autoregressive models could also be used.

As in temporal modeling, each sensor node i has its own model h_i, which is updated when the prediction is above the user-defined error threshold ϵ. The copy of the model is however not maintained by the base station, but by the parent node in the routing tree. Each node j that has children maintains a model h_{j-i} for each of its children i. A copy of these models is maintained by the base station. A second user defined error threshold ϵ_Δ is used to determine when to update these models. The way models are distributed in a network is illustrated in Figure 3.6, for a network of four nodes and a routing tree of depth three. Models produced by nodes are listed above each node, and models used to get predictions are listed below each nodes.

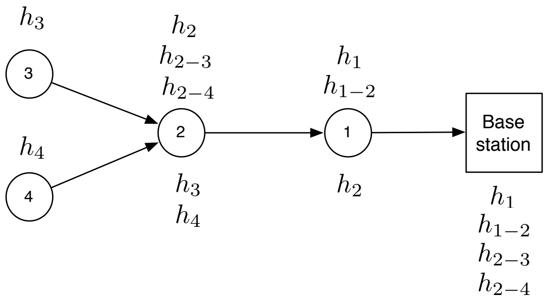


Figure 3.6: Replicated models with edge monitoring: the models h_i are used to infer sensor measurements, while models h_{j-i} are used to monitor the differences between the measurements of two adjacent nodes.

The measurement of a node i is obtained by the base station by summing the prediction for the root node measurement, and the predictions for the differences between all pairs of nodes that separate node i from the base station. For example, in the network illustrated in Figure 3.6, a prediction for sensor 4 is obtained by summing

$$\hat{s}_4[t] = \hat{s}_1[t] + \hat{\Delta}_{1-2}[t] + \hat{\Delta}_{2-4}[t]$$

Since $\Delta_{j-i}[t] = s_j[t] - \hat{s}_i[t]$, and that the RM scheme ensures $|\hat{s}_i[t] - s_i[t]| < \epsilon$, that $|\hat{\Delta}_{j-i}[t] - \Delta_{j-i}[t]| < \epsilon_\Delta$, it follows that

- a prediction for the root node can be obtained with an accuracy $\pm\epsilon$
- a prediction for a node l hops away from the root node can be obtained with an accuracy of $\pm(\epsilon + l(\epsilon + \epsilon_\Delta))$.

The accuracy therefore depends on the depth of a node in the routing tree, and lower ϵ values must therefore be used to achieve the same accuracy as in temporal modeling.

Clique models: The rationale of clique models, investigated in [15], is to partition the network into groups of sensors, called *cliques*, with one replicated model for each clique. Note that the term clique here refers to group of nodes, and is not related to the notion of clique in graph theory. The measurements of sensors in a clique are gathered at a common sensor node, called clique root. The clique root may not be part of the clique. Let $\{\mathcal{S}_k\}_{1\leq k\leq K}$ be a partitioning of $\mathcal{S}$ in K cliques, i.e., a set of K cliques such that $\mathcal{S} = \cup_{1\leq k\leq K}\mathcal{S}_k$. Let $i_k^{root} \in \mathcal{S}_k$ be the sensor node that takes the role of the root of clique k.

The measurements of the set of sensors $i \in \mathcal{S}_k$ are gathered at the clique root i_k^{root}, where data is modeled by a model $h_k(x, \theta)$. Inputs x may take values in the set of measurements of the sensors in the clique. The clique root then transmits to the sink the minimal subset of parameters/data such that the measurements $s_i[t]$ and model counterpart $\hat{s}_i[t]$ do not differ more by than ϵ. An illustration of a clique partition is given in Figure 3.7. Note that updating the model may require a clique root to rely on multi-hop routing. For example, sensor node 3 may not be in communication range of the base station, and may require some nodes of $\mathcal{S}_2$ to relay its data.

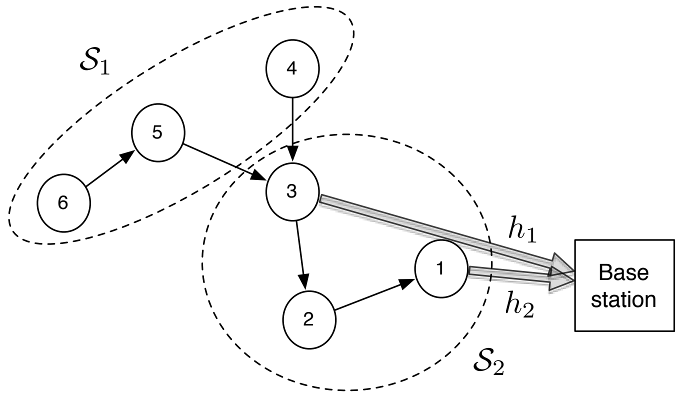


Figure 3.7: Clique models: the network is here partitioned in two cliques $\mathcal{S}_1 = \{4, 5, 6\}$ and $\mathcal{S}_2 = \{1, 2, 3\}$, with replicated models h_1 and h_2 for each clique. The clique root of $\mathcal{S}_1$ is sensor node 3 and the clique root of $\mathcal{S}_2$ is sensor node 1.

The choice of the model and the partitioning of cliques are clearly central pieces of the system. Inspired by the work of [19] concerning model-driven data acquisition, authors in [15] suggest to use gaussian models. The clique root collects data from all the sensors in the clique for N epochs, compute the mean vector and covariance matrix, and communicate these data to the base station. Then, at every epoch, the clique root determines what measurements must be sent so that the base station can infer the missing measurements with a user defined ϵ error threshold.

The goal of the approach is to reduce the communication costs. These are divided into intra-source and source-base station costs. The former is the cost incurred in the process of collecting data by the clique root to check if the predictions are correct. The latter is the cost incurred while sending the set of measurements to the sink. Authors show that the problem of finding optimal cliques is NP-hard, and propose the use of a centralized greedy algorithm to solve the problem. The heuristic starts by considering a set of S cliques, i.e., one for each sensor, and then assesses the reduction of communication obtained by fusing all combinations of cliques. The algorithm stops once fusion leads to higher communication costs.

Discussion

Replicated models have a high potential in reducing communications, and their main advantage is to guarantee bounded approximation errors. Temporal modeling is easy to implement, and the different proposed approaches do not require much computation, which makes them suitable for the limited computational resources of sensor nodes. In terms of communications savings, the network load of sensors is reduced by a factor proportional to the number of updates required to maintain synchronized the models between the sensors and the base station. The highest network load is sustained by the base station, and depends on the number of updates sent during an epoch. If all the sensor measurements can be predicted, then no update is sent and the load is therefore null for all sensors. At the other extreme, if all sensor nodes send an update, the load distribution is similar to that of collecting all the measurements. Therefore, replicated models give

$$L_{\max} \sim O(S).$$

The modeling of spatial dependencies is attractive as spatial correlations are very often observed in sensor network data. Also, it is likely that temporal models all have to update their parameters at about the same time, i.e., when an unexpected event occurs in the environment for example. The modeling of spatial dependencies however raises a number of concerns. In the edge monitoring approach, the error tolerance ϵ must be reduced in order to provide the same accuracy guarantee as in temporal modeling approaches [80]. In clique models, the partitioning of the network is a computationally intensive task that, even if undertaken by the base station, raises scalability issues [15]. In terms of communication savings, the network load of sensors is reduced, as for temporal modeling, by a factor that depends on the average number of updates. The savings can however be much higher, particularly, in situations where all measurements increase by the same amount for example.

An issue common to all the approaches based on replicated models is packet losses. In absence of notification from a sensor node, the base station deems the prediction given by the shared model to fall within the ϵ error tolerance. Additional checking procedures must therefore be considered for this scheme to be completely reliable. To that end, a "watchdog" regularly checking the sensor activity and the number of sent packets can be set up, as discussed in [79] for example. By keeping the recent history of sent updates in the sensor node memory, these can be communicated to the sink at checking intervals if the number of sent packets differ from the number of received packets. Node failure is detected by absence of acknowledgment from the sensor node to the watchdog request. Finally, the choice of the model is also an important factor in the efficiency of RM. This issue will be the subject of Chapter 4.

3.4 Aggregative approaches

Aggregative approaches are based on aggregation services such as TAG, presented in Section 2.1.4. Data aggregation services allow to aggregate sensor data in a time- and energy-efficient manner, by synchronizing the activities of the sensor nodes as a function of their depth in the routing tree. Besides reducing the sensor nodes' energy consumption, aggregation services also allow to combine data as they flow to the base station.

Using the terminology of [56, 58], an aggregate of data is called a *partial state record* and is denoted by $\langle . \rangle$. It can be any data structure, such as a scalar, a vector or a matrix for example. Partial state records are initialized locally on all nodes, and then communicated and merged in the network. When the partial state record is eventually delivered by the root node to the base station, its elements may be recombined in order to provide the observer with the final output. Methods based on aggregation require the definition of three primitives [56, 58]:

- an initializer *init* which creates a partial state record,
- an aggregation operator f which merges partial state records, and
- an evaluator e which returns, on the basis of the partial state record finally delivered to the base station, the result required by the application.

The partial state records are merged from the leaf nodes to the root, along a synchronized routing tree such as presented in Section 2.1.4.

In practice: Aggregation services were first shown [56, 58] to be able to compute simple operations like the minimum, the maximum, the sum or the average of a set of measurements. For example, for computing the sum, the following primitives can be used:

$$\left\{ \begin{array}{rcl} init(s_i[t]) & = & \langle s_i[t] \rangle \\ f(\langle S1 \rangle, \langle S2 \rangle) & = & \langle S1 + S2 \rangle \\ e(\langle S \rangle) & = & S. \end{array} \right.$$

Measurements are simply added as they are forwarded along the routing tree. The resulting aggregate obtained at the base station is the overall sum of the measurements. The main advantage of aggregation is that the result of an operator is computed without collecting all the measurements at the base station. This can considerably reduce the amount of communication.

In data modeling, aggregation services can be used to compute the parameters of models. For example, let us consider a sensor network monitoring the temperature in a room where an air conditioning system is set at $20°C$. Most of the time, the temperature measurements of sensor nodes are similar, and can be approximated by their average measurements. The *average model* [38, 56, 15] was one of the first proposed, as its implementation is fairly straightforward. The interest of such a model can be to detect, for example, when a window or a door is opened. To do so, the average of the measurements is first computed by means of an aggregation service and retrieved at the base station. The result is then transmitted back to the sensor nodes, which can compare their measurements with average measurement. If the difference is higher than some user-defined threshold, a sensor node can notify the base station that its local measurement is not in agreement with the average measurement.

In [30], authors showed that the coefficient of a linear model can be computed using aggregation services. Their approach, called *distributed regression*, allows to use more complex models for representing sensor measurements as a function of spatial coordinates. The average model is first presented in the following, and is followed by the presentation of the distributed regression algorithm.

Average model

The average model is a simple model that well illustrates the rationale of aggregation. It consists in modeling all the sensor measurements by their average value, i.e.

$$\hat{s}_i[t] = \mu[t]$$

where $\mu[t] = \frac{\sum_{i=1}^{S} s_i[t]}{S}$ is the average of all the sensor measurements at epoch t [38, 56, 15]. The average is obtained by the following primitives:

$$\left\{ \begin{array}{rcl} init(s_i[t]) & = & \langle 1, s_i[t] \rangle \\ f(\langle C1, S1 \rangle, \langle C2, S2 \rangle) & = & \langle C1 + C2, S1 + S2 \rangle \\ e(\langle C, S \rangle) & = & \frac{S}{C}. \end{array} \right.$$

The partial states record $\langle C, S \rangle$ is a vector of two elements, consisting of the count and the sum of sensor measurements. The aggregation process is illustrated in Figure 3.8. This way, only two pieces of data are transmitted by all sensor nodes. The main advantage of aggregation is that all nodes send exactly the same amount of data, which is independent of the number of sensors. It therefore provides a scalable way to extract information from a network. Also, it may dramatically decrease the highest network load, typically sustained by the root node. In the network of nine sensors represented in Figure 3.8, the transmission of all measurements to the base station would cause the root node to send nine measurements at every epoch. This number is reduced to two thanks to the aggregation process.

Once obtained at the base station, aggregates can be transmitted from the base station to sensor nodes [30]. This allows sensor nodes to compare their measurement to the average measurement, and to provide the base station with their measurement if

$$|s_i[t] - \mu| > \epsilon$$

where ϵ is a user defined error threshold.

This is illustrated in Figure 3.9, where nodes 1 and 8 actually send their true measurement after receving the feedback $\mu[t]$ from the base station. Such a strategy allows to bound the approximation errors of the average model. It however implies additional communication rounds between the base station and the sensor nodes, which is intuitively expensive in terms of communication.

Distributed regression

Similarly to the average model, an aggregative approach can be used to compute the regression coefficients of a linear model. The approach was investigated by Guestrin et al. in [30], who relied on basis functions to approximate sensor network measurements (cf. 2.2.3). The basis functions may be defined over space and time, allowing in some cases to compactly represent the overall spatiotemporal variations by a small set of coefficients.

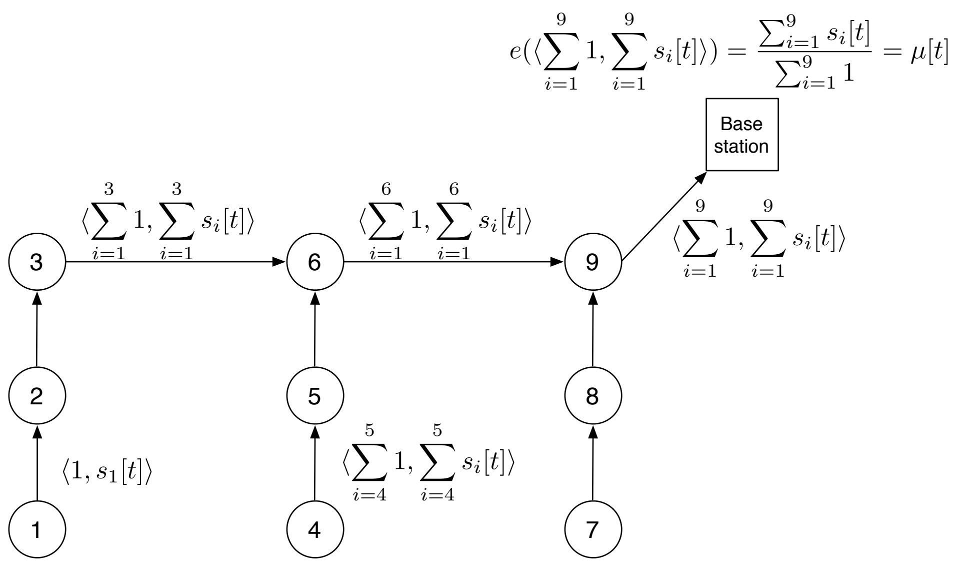

$e(\langle\sum_{i=1}^{9} 1, \sum_{i=1}^{9} s_i[t]\rangle) = \frac{\sum_{i=1}^{9} s_i[t]}{\sum_{i=1}^{9} 1} = \mu[t]$
Base station
$\langle\sum_{i=1}^{3} 1, \sum_{i=1}^{3} s_i[t]\rangle$
$\langle\sum_{i=1}^{6} 1, \sum_{i=1}^{6} s_i[t]\rangle$
$\langle\sum_{i=1}^{9} 1, \sum_{i=1}^{9} s_i[t]\rangle$
$\langle 1, s_1[t]\rangle$
$\langle\sum_{i=4}^{5} 1, \sum_{i=4}^{5} s_i[t]\rangle$
1
2
3
4
5
6
7
8
9

Figure 3.8: Aggregation service at work for computing the average $\mu[t] = \frac{\sum_{i=1}^{9} s_i[t]}{\sum_{i=1}^{9} 1}$ of the measurements taken by sensors at epoch t.

More precisely, let $\mathcal{H} = \{\pi_1, \ldots, \pi_p\}$ be a set of p basis functions which are used to represent the sensor measurements. This set must be defined by the observer, prior to running the algorithm. The inputs of these basis functions can be functions of the time t or of the sensor coordinates $c = (c_1, c_2, c_3)$ (assuming 3D coordinates) for example. Let p be the overall number of basis functions and let θ_j be the coefficient of the j-th basis function. The approximation to a sensor measurement at time t and location c is given by:

$$\hat{s}(c,t) = \sum_{j=1}^{p} \theta_j \pi_j(c,t)$$

Example: A quadratic regression model over time $\hat{s}(c,t) = \theta_1 t + \theta_2 t^2$ is defined by two basis functions $\pi_1(c,t) = t$ and $\pi_2(c,t) = t^2$. The addition of $\pi_3(c,t) = c_1$ and $\pi_4(c,t) = c_2$ gives a model that captures correlations over space. An intercept can be added with $\pi_5(c,t) = 1$.

Using the notations defined before, we have a model h which represents the overall variations in the sensor field by

$$\begin{aligned} h_\theta : \mathbb{R}^p &\rightarrow \mathbb{R} \\ x &\mapsto \hat{s}(c,t) = \sum_{j=1}^{p} \theta_j \pi_j(c,t) \end{aligned}$$

where the inputs $x = (\pi_1(c,t), \ldots, \pi_p(c,t))$ are functions of time and coordinates, and the parameters $\theta = (\theta_1, \ldots, \theta_p)$ are the coefficients of the linear model. Approximations

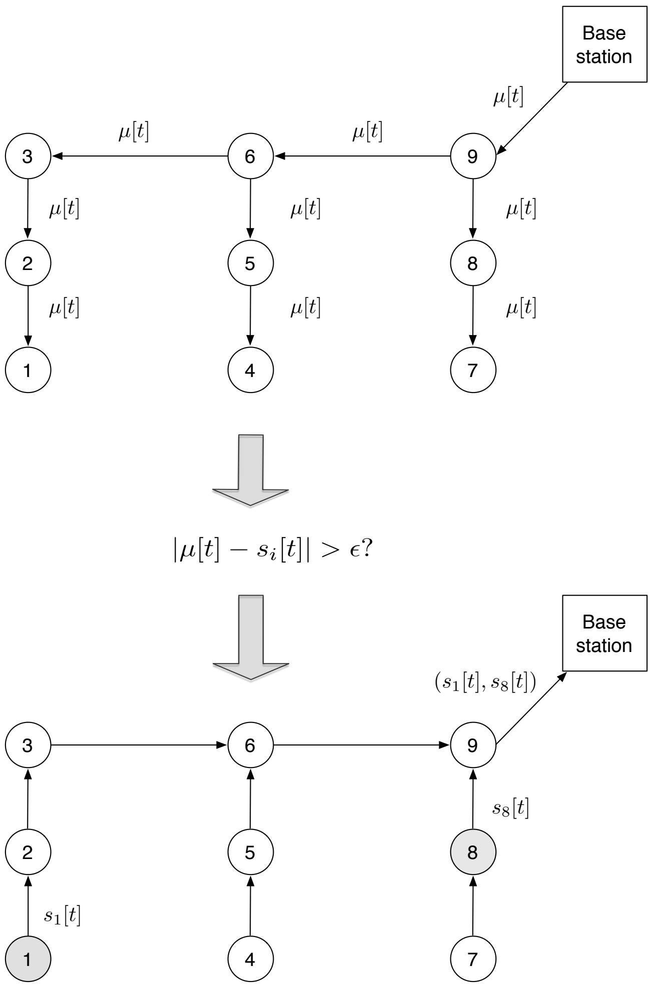


Figure 3.9: The aggregate $\mu[t]$ can be communicated to all sensors, allowing them to compute locally $|s_i[t] - \mu[t]|$, and to send their true measurement $s_i[t]$ is the difference $|s_i[t] - \mu[t]|$ is higher than a user defined error threshold ϵ. In this example, sensors 1 and 8 update their measurements.

$\hat{s}_i[t]$ for sensor i are given by specifying the coordinates c_i of sensor in the model, i.e., $\hat{s}_i[t] = \sum_{j=1}^{p} \theta_j \pi_j(c_i, t)$. An interesting feature of this model is that it not only provides approximations to sensor measurements, but also allows to provide predictions for all locations in the field.

Assuming that N_i measurements $s_i[t]$ have been taken at locations c_i for N_i epochs t, and let $N = \sum_{i=1}^{S} N_i$ be the overall number of measurements taken by sensor nodes. The coefficients θ_j can be identified by minimizing the mean squared error between the actual and approximated measurements (standard linear regression procedure, cf. Section 2.2.3). Let Y be the $N \times 1$ matrix that contains these measurements, and let θ be the column vector of length p which contains the coefficients θ_j. Finally, let X be the $N \times p$ matrix whose columns contain the values of the basis functions for each observation in Y. Using this notation, the optimization problem can be stated as

$$\theta^* = \arg\min_{\theta} ||X\theta - Y||^2$$

which is the standard optimization problem in regression (cf. Section 2.2.3). The optimal coefficients are found by setting the gradient of this quadratic objective function to zero, which implies

$$(X^T X)\theta^* = X^T Y. \tag{3.13}$$

Let $A = X^T X$ and $b = X^T Y$. A is referred to as the scalar product matrix, and b as the projected measurement vector. In distributed regression, the measurements $s_i[t]$ do not need to be transmitted to the base station. Instead, the matrix A and vector b are computed by the aggregation service. Once aggregated, the coefficients θ can be computed at the base station by solving Equation 3.13.

Let X_i be the $N_i \times p$ matrix containing the values of the basis functions for sensor i, and let Y_i be the $N_i \times 1$ matrix that contains the measurements taken by sensor i. Both X_i and Y_i are available at sensor i, which can therefore compute locally $A_i = X_i^T X_i$ and $b_i = X_i^T Y_i$.

The matrix A and the vector b are actually sums of $A_i = X_i^T X_i$ and $b_i = X_i^T Y_i$. Using this fact, A and b can be computed by an aggregation service by merging along the routing tree the contributions A_i and b_i of each sensor.

Indeed, assuming S sensors, each of which collects a measurement $s_i[t]$, we have

$$a_{j,j'} = \sum_{i=1}^{S} \pi_j(c_i, t)\pi_{j'}(c_i, t) \tag{3.14}$$

where $a_{j,j'}$ if the entry at the j-th row, j'-th column of the scalar product matrix A. Similarly, we have

$$b_j = \sum_{i=1}^{S} \pi_j(c_i, t)s_i[t] \tag{3.15}$$

where b_j if the j-th element of the projected measurement vector. All the elements $a_{j,j'}$ and b_j can be computed by means of an aggregation service, using the following primitives:

- For elements $a_{j,j'}$:

$$\begin{cases} init(\text{i}) & = & \langle \pi_j(c_i, t)\pi_{j'}(c_i, t) \rangle \\ f(\langle S1 \rangle, \langle S2 \rangle) & = & \langle S1 + S2 \rangle \\ e(\langle S \rangle) & = & S \end{cases}$$

- For elements b_j:

$$\begin{cases} init(\text{i}) & = & \langle \pi'_j(c_i, t)s_i[t] \rangle \\ f(\langle S1 \rangle, \langle S2 \rangle) & = & \langle S1 + S2 \rangle \\ e(\langle S \rangle) & = & S \end{cases}$$

The computation of the matrix A requires the aggregation of p^2 elements, while the aggregation of the vector b requires the aggregation of p elements. Once all the elements are retrieved at the base station, the set of coefficients θ can be computed by solving the system

$$A\theta = b.$$

The resulting model allows to get approximations to sensor measurements. Depending on the model, approximations may also be obtained for other spatial coordinates or at future time instants (cf. example above). As with the average model, the parameters θ may be communicated to all nodes, allowing each sensor to locally compute the approximation obtained at the base station. This makes it possible to check that approximations are within an error threshold ϵ defined by the observer. All sensors whose approximations differ by more than $\pm\epsilon$ may notify their true measurements to the base station.

Discussion

The main advantage of aggregative approaches is that they allow to represent the variations of sensor measurements by means of models whose number of coefficients is independent of the number of sensor nodes in the network. This makes these approaches scalable to large networks. Furthermore, they allow to evenly distribute the number of radio transmissions among sensor nodes. Compared to model-driven and replicated models approaches, the network load of leaf nodes is increased, whereas the load of nodes close to the base station is reduced. Considering that the highest network load primarily determines the network lifetime, aggregative approaches are particularly attractive for reducing the load of sensor nodes close to the base station.

The highest network load depends on whether bounded approximation errors are required. Retrieving the model coefficients causes a highest network load of

$$L_{\max} \sim O(p^2)$$

where p is the number of parameters of the model. If bounded approximation errors are required, the p coefficients must be communicated to all nodes, causing p transmissions. Depending on the number of sensors for which approximations are more than ϵ away from their true measurements, an additional number of updates of up to S may be sent. Denoting by $L_{\max}^{\text{check}}$ the highest network load when approximations are checked against the true measurements, we have

$$L_{\max}^{\text{check}} \sim O(p^2 + S).$$

The upper bound is higher than for data collection where all measurements are collected, and for which we had $L_{\max} = O(S)$ (cf. Section 3.1). The distributed regression with bounded errors therefore may lead to higher communication costs if the model does not properly reflect the variations of sensor measurements.

The choice of the model is thus an important issue, particularly when there is no a priori information on the type of variations. In practice, a solution may be to collect data from the whole network in order to get an overview of the types of measurement patterns. On the basis of this initial stage, different models may be tried and assessed at the base station, in order to select a model that properly fits the data.

It is worth noting that different optimizations can be brought to these approaches. In particular, the elements of the matrix A do not depend on sensor measurements. In the case of spatial models, they only depend on the spatial coordinates of the sensors. If these coordinates are known by the base station, the matrix A may be computed straightaway at the base station, thus saving $0(p^2)$ transmissions. In the same way, the base station may also infer the entries of A when time is involved. The load can therefore be reduced to $O(p)$ if no error threshold is set.

3.5 Summary

This chapter provided a state of the art on the use of learning techniques for reducing the amount of communication in sensor networks. Classifying these approaches in three main types, namely model driven, replicated models, and aggregative approaches, we outlined for each of them their strengths and their limits. Table 3.1 gives a summary of the different learning schemes in terms of error type and highest network load.

Learning scheme	Error type	Highest network load
Model-driven acquisition	Probabilistic bounded or unbounded	$L_{\max}^{MD} \sim O(\|\mathcal{S}_q\|)$
Replicated models	ϵ-bounded	$L_{\max}^{RM} \sim O(S)$
Aggregative approaches	Unbounded or ϵ-bounded	$L_{\max}^{DR} \sim O(p^2)$ $L_{\max}^{DR_{\text{check}}} \sim O(p^2 + S)$

Table 3.1: Comparison of the performances of the different modeling approaches presented in this chapter.

- Approaches based on **model-driven acquisition** reduce the highest network load to $|\mathcal{S}_q|$, i.e., the number of sensors whose measurements are effectively collected. The main characteristic of these approaches is that part of the network can remain in the idle mode. Model-driven data-acquisition therefore not only reduces the highest network load, but also allows to reduce to a negligible level the energy consumption of the sensor nodes not queried. In the idle mode, the energy consumption is about four orders of magnitude lower than in the active mode (see Table 2.2 - for example $5\mu A$ in the idle mode against $19.5mA$ with MCU active for the Telos node).

  The subset of sensor nodes whose measurements are collected could be changed over time to distribute the energy consumption. Indeed, there exists in most cases different pairs of set of queried and predicted sensors for which the observer's accuracy requirements can be satisfied. Further work should be done in this direction.

  The error type entailed in the obtained predictions depends on whether the model can be trusted. If the model is correct, predictions can be bounded with an error threshold ϵ and a confidence level $1 - \delta$. The drawback of model-driven approaches is however that unexpected events in the monitored phenomenon may not be detected if they concern locations where measurements are predicted. The use of multiple pairs of sets of queried and predicted sensors can be used to address this issue. Indeed, assuming that all sensor nodes will at some point be part of the set of queried sensors, an unexpected event will be detected when the sensor nodes monitoring the location of the event are queried. The time elapsed before the event is detected depends on the frequency at which subsets of sensors are changed. This time may be long, and therefore model-driven approaches are not well-suited to event detection tasks.

- The main characteristic of **replicated models** is to guarantee ϵ-bounded prediction errors, even in the case of unexpected events. These approaches however require the sensor nodes to take their measurements at every epoch, so that they can be compared with the predicted measurements. The energy savings depend on the frequency of updates of the model. In the optimal case, the predictions obtained by the model are always within $\pm\epsilon$ of the true measurements, and therefore no communication is needed. The energy savings are in this case around one order of magnitude (see Table 2.2 - $1.8mA$ in the idle mode against $19.5mA$ with MCU active for the Telos node).

  In the worst case, updates are needed at every epoch. The HNL is in $O(S)$ for this worst scenario, and has therefore the same order of magnitude than the default data collection scheme. Depending on the number of parameters sent in each model update, the exact amount of communication can even be higher with replicated models than for the default data collection scheme.

  The efficiency of replicated models to reduce communication therefore depends on the prediction model used. This issue will be discussed extensively in Chapter 4, which will present our first contribution in this thesis.

- Finally, **aggregative approaches** lead to either unbounded or ϵ-bounded error. The type of error depends on whether aggregates obtained at the base station are communicated to sensor nodes. If no check is made against the true measurement, the highest network load is reduced to the number of aggregates collected, i.e., $O(p^2)$, where p is the number of parameters in a model. It is worth noting that optimization techniques can be used to reduce the HNL in $O(p)$. The main characteristic of aggregative approaches with unbounded errors is that the HNL does not depend on the number of sensors. This makes these approaches scalable to large network.

  The model coefficients computed at the base station can be communicated to sensor nodes. This allows sensor nodes to compute locally the predictions obtained at the base station, enabling aggregative approaches to deal with event detection. The use of this feature can however be expensive in terms of communications, as an additional network load in $O(p + S)$ can be reached in the worst case.

In summary, modeling techniques can greatly reduce energy consumption, up to several orders of magnitude with model-driven approaches. The use of models however implies some approximations of the sensor measurements. For an observer, it is often important that the approximation errors are bounded, i.e., within $\pm\epsilon$ of the true measurements. This guarantee is only possible with replicated models and aggregative approaches, which allow to compare the model predictions with the true measurements.

The efficiency of modeling techniques in reducing communication mainly depends on the model chosen. If the prediction models used are adapted to the type of measurements collected by sensors, high communication savings can be obtained. In the case where the models are not adapted, modeling approaches can however lead to worst case scenario where the energy consumption is higher than in the default data collection scheme.

The selection of an efficient model is therefore a primary issue. The following chapter presents our first contribution, aimed at dealing with this issue for temporal replicated models.

Chapter 4

Adaptive Model Selection

This chapter presents our first contribution in this thesis, called *Adaptive Model Selection* (AMS), which extends previous work on replicated model approaches [72, 47, 41, 87, 79]. The design of the AMS algorithm is motivated by the fact that the identification of prediction models that effectively reduce communication is in practice a difficult task. In particular, the use of models that are not adapted to the sensor measurements can lead to an increase of communication. One of the main reasons is that in most cases, very little or no *a priori* information is available on the dynamic of the sensor measurements.

Section 4.1 discusses the conditions allowing a model to reduce the communication costs. Section 4.2 presents the AMS algorithm. Its rationale is to let a wireless node run different prediction models, and to select over time the one that provides the most savings in terms of network load. Section 4.3 reviews different strategies for time series prediction. Section 4.4 provides an experimental analysis of the different approaches on a set of fourteen different time series.

4.1 Which model to choose?

Let us consider the following example in order to introduce the topic. Figure 4.1 illustrates the use of a constant model for approximating a set of temperature measurement collected over a one-day period in the greenhouse of the Solbosch, during one of the data collection experiments we made. The accuracy is set to $2°C$. Using a constant model [72], the percentage of updates of the sensor is of 8% compared to a periodic data collection where measurements are sent at every epoch. We note that the amount of communication savings is already considerable.

Looking more closely at the figure, we see that the number of updates is more important during the day time, particularly because the constant model is not appropriate for modeling the increase or the decrease of the temperature measurements. These temporal variations are better captured by a model that takes into account the time, such as an autoregressive model [87, 79]. We illustrate this in Figure 4.1, right, by reporting the approximations provided by an autoregressive model of order two. The models were computed using the last sixty measurements, as suggested in [87]. An AR(2) allows to better capture the temporal variations, and reduces the percentage of updates to 6%.

More complex models, such as autoregressive models, can therefore better model the variations of the measurements taken by sensors. The ability of these models to better approximate sensor measurements however depends on a number of parameters, such as

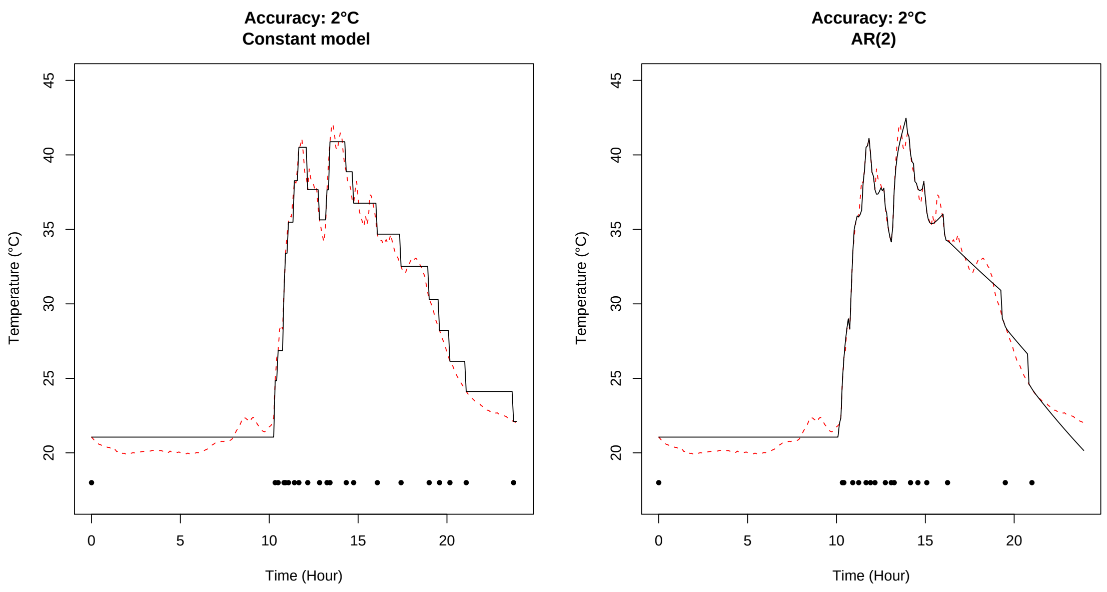
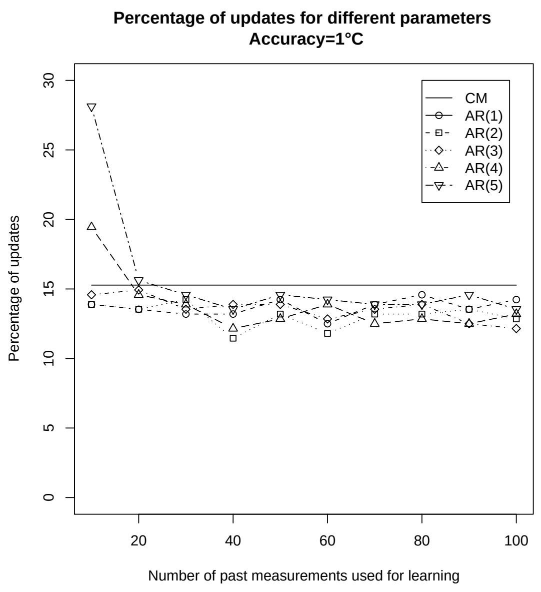


Figure 4.1: Approximations obtained using replicated constant (left plot) and AR(2) (right plot) models on a one-day temperature data collection in the Solbosch greenhouse. Updates are reported with black dots on the bottom of the plots. The use of an autoregressive model, which takes the temporal variations into account, allows to reduce the number of updates by two percent.

the order of the model for autoregressive models. The number of training examples for learning the model parameters is also an important factor. Failure to choose the right model parameters can lead to poor prediction accuracies. Last, but not least, the nature of the time series and the error tolerance ϵ required by the observer play an important role.

This is illustrated in Figure 4.2 where we report the percentage of updates obtained by using constant model (CM), and autoregressive models of order p (AR(p)), with p ranging from 1 to 5. The number of training examples is varied from 10 to 100 by steps of 10. The results are reported for error tolerance of $\epsilon = 1°C$ (left), and $\epsilon = 5°C$ (right). Previous work on replicated models was mainly driven by the fact that more complex models *could* provide higher communication savings [41, 87, 79]. Few guidelines were however provided regarding the choice of the parameters. As seen in Figure 4.2, *the choice of the parameters is of primary importance.* For example, although autoregressive models provide in most cases less updates than the constant model when the error tolerance is of $1°C$ (Figure 4.2, left), the converse is observed when the error tolerance is of $5°C$ (Figure 4.2, right). The number of past measurements used to train the model is also an important parameter. Less than 20 measurements for instance lead autoregressive models with order higher than three to imply a number of updates higher than the constant model. We note that this is a result of the bias/variance tradeoff discussed in Section 2.2.2.

In qualitative terms, these results show that constant model outperforms autoregressive models when the error tolerance is high, or when the number of measurements used for the learning is not large enough. These experimental observations are generally valid, as will be seen in Section 4.4 where the performances of replicated models on different types of time series are analyzed. It is however difficult, *a priori*, to provide guidelines for the choices of



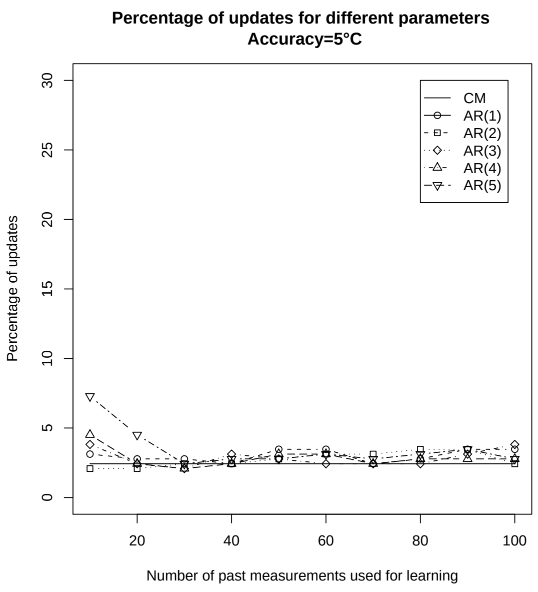


Figure 4.2: Percentage of updates caused by models of increasing complexity, for $\epsilon = 1°C$ (left) and $\epsilon = 5°C$ (right) error thresholds. The benefits of complex models diminishes when the error threshold is relaxed.

the parameters related to the model and the learning procedure.

A second issue that was overlooked in the previous work is that the size of an update must also be taken into account. In previous work [72, 40, 87, 79], the considered metric was the *update rate*, which is the percentage of updates between a wireless node and the base station over time. Denoting by $I[t]$ the indicator function characterizing instants of update, i.e.,

$$\begin{aligned} I : \mathcal{T} &\rightarrow \{0,1\} \\ t &\mapsto \begin{cases} 0 & \text{if } |\hat{\mathrm{s}}[\mathrm{t}] - \mathrm{s}[\mathrm{t}]| \leq \epsilon \\ 1 & \text{if } |\hat{\mathrm{s}}[\mathrm{t}] - \mathrm{s}[\mathrm{t}]| > \epsilon \end{cases} \end{aligned} \tag{4.1}$$

the update rate can be expressed as

$$U[t] = 100 \frac{\sum_{\tau=1}^{t} I[\tau]}{t}.$$

The update rate for the periodic data collection, also referred to as *default monitoring scheme* in the following, is 100% since the base station is updated with a measurement at each epoch t. Any lower value indicates a gain in the number of transmitted packets.

A deficiency of the metric is that it does not take into account the size of model updates, which typically increases as the number of parameters of a model gets higher. We therefore introduce an alternative metric, the *weighted update rate*, that takes into account the size of a model update. We define it as

$$W[t] = U[t]C \tag{4.2}$$

where C, henceforth referred to as *model weight*, quantifies the relative communication costs of transmitting a model update in comparison to the default scheme. We will consider that in the default scheme, the communication of one measurement implies the transmission of a packet which contains the measurements, and that the communication of a model update implies the transmission of a packet which contains the parameters necessary for the base station to run the model.

In order to fix an idea of this ratio, the following quantities can be for example considered. In the packets sent in TinyOS [85], a reference operating system for sensor networks, the packet overhead (additional bytes of information such as synchronization data and sensor ID) typically ranges between 12 and 36 bytes. Assuming that the update of an AR(p) model requires $2p$ bytes (p bytes for the last p measurements and p bytes for the parameters), and that the packet overhead is on average of 24 bytes, the model weight of an AR(p) model is therefore of

$$C_{AR(p)} = \frac{24 + 2p}{24 + 1}. \tag{4.3}$$

For the constant model (CM), as an update only consists of one measurement, the model weight is $C_{CM} = 1$. The integration of the model weight in the metric used to assess the communication savings achievable by replicated models allows to make the two following observations. First, models that rely on parameters all have a model weight higher than one. Therefore, these models may lead to communication costs higher than the default scheme. Second, the constant model, by having a model weight of 1, always ensures that the communication costs related to its use are not greater than the default scheme.

4.2 Adaptive Model Selection

This section presents the adaptive model selection (AMS), which aims at allowing a sensor node to determine autonomously what prediction model to use. The rationale of our approach is to let the sensor node compare the ability of different models to predict its measurements, and to select over time the model that provides the lowest weighted update rate. A model selection strategy called *racing* is used to discard over time the models that perform poorly.

Overview

As emphasized in the previous section, the constant model is the a priori best option in replicated models for situations where no information is available about the type of measurements that a sensor node will collect. The constant model is however basic, and the use of more complex models should be considered given that they can lead to higher communication savings. In AMS, we suggest to initially provide the sensor nodes with a collection of K models

$$\{h_k(x, \theta_k)\},\ 1 \leq k \leq K.$$

Their performances in terms of weighted update rate, denoted by $W_k[t]$, $1 \leq k \leq K$, are compared over time by the sensor node.

A first issue consists in choosing the type and number of prediction models to include in the collection $\{h_k\}$. An extreme situation consists in considering only one model. In this case, in the absence of a priori information on the signal, it is clear that the constant model should be used. The addition of more complex models to the collection should be considered by keeping in mind that the computational resources of a wireless sensor are limited. Among existing techniques in time series forecasting, autoregressive models are suitable candidates, as was motivated in previous work [87, 79]. The collection of K autoregressive models can be built by considering models with orders from 1 to K for example. The performances $W_k[t]$ at time t for each model are assessed by a sensor node by using the following recursive formulation

$$W_k[t] = \frac{(t-1)W_k[t-1] + I[t]}{t} \tag{4.4}$$

where $I[t]$ is the indicator function defined in Equation (4.1).

In the collection of models, only one is shared with the base station, called the *current* model. This model is the one assumed to have the best performances, and is denoted h_{best}. When starting the AMS, the constant model is assumed to be the best. Whenever an update is required because the current model fails to predict within ϵ the current measurement, the new model sent to the sink is the one that provides the best performances, i.e., the model h_k such that

$$k = \arg\min_k W_k[t].$$

Over time, the weighted update rate $W_k[t]$ of a model converges to a value that gives the average performance of the model. After a sufficient amount of time, one of the models in the collection of models will prove to have the best performance in terms of communication savings for modeling the sensor measurements. In order to reduce the computational effort of the sensor nodes, the models that have lower performances can be discarded. The decision of when to stop maintaining a model which has lower performances that another can be addressed by means of the *racing algorithm*.

Racing algorithm

The racing algorithm is a model selection technique originally proposed in [64]. Given a collection of K models and N observations, the rationale of the racing is to assess the generalization of the models in a competitive fashion, with at most N steps. The k-th step consists in estimating the parameters of each model using all but the k-th observation, and in computing the prediction error on the k-th observation.

After N steps, the procedure comes down to a leave-one-out validation (Section 2.2.2). After only a small number of tests, it is however usually possible to distinguish the best models from the worst models. This is done using statistical bounds, which allow to determine how close the estimated performance of a model is to its true performance. The models which are significantly worse than the best ones are thrown out of the race and not tested again [64, 65].

The racing algorithm is well adapted to our scenario, as the prediction error on unseen samples is possible every time a new measurement is collected. The performance metric is the weighted update rate, and the problem therefore consists in determining, for a model k, a confidence interval of the estimate of $W_k[t]$ at time t. Figure 4.3 illustrates the strategy, where six models are assessed in parallel. The estimated performances are given by the black dots, and the confidence intervals by the vertical lines surrounding the estimated performances. The models which have performances whose lower confidence bound is higher than the upper confidence bound of the best model can be discarded.

Different statistical tests exist to estimate the confidence bounds of an estimate, which depend on the assumptions that can be made on the probabilistic distribution of the estimated performances. For example, if the distribution is known to be Gaussian, then a student t-test could be used. In general, it is difficult to formulate assumptions on the na-

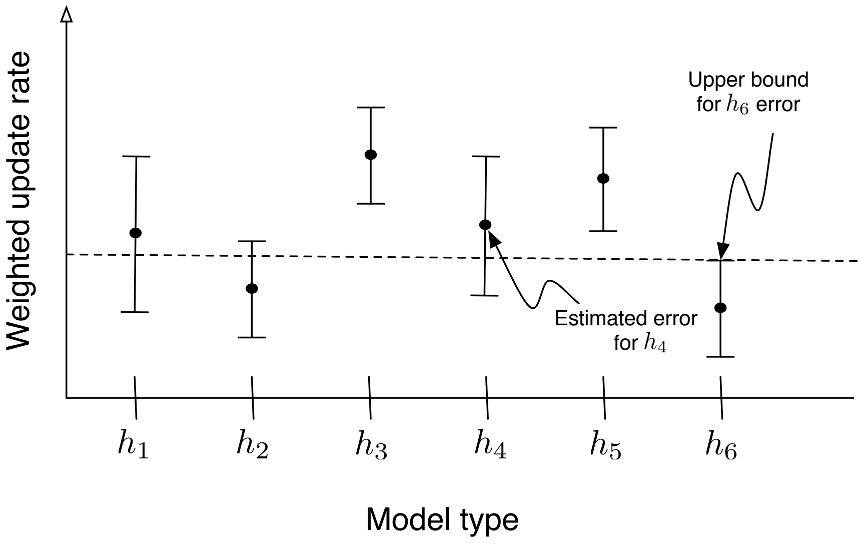


Figure 4.3: Confidence intervals associated to the model performances estimates. Models h_3 and h_5 are outperformed by model h_6.

ture of this distribution, and the Hoeffding bound can be used [65, 35]. This bound states that, with probability $1 - \delta$, h_{best} truly outperforms h_k if

$$\Delta_{h_k,h_{\text{best}}}[t] > R\sqrt{\frac{\ln(1/\delta)}{2t}}, \tag{4.5}$$

where R is the range taken by the difference $\Delta_{h_k,h_{\text{best}}}[t] = |W_k[t] - W_{\text{best}}[t]|$. Thanks to the lack of parametric assumptions, the Hoeffding bound requires no other information than the range of values taken by quantities considered, which is known in advance. Indeed, as $0 \leq W_k[t] \leq C_k$ and $0 \leq W_{\text{best}}[t] \leq C_{\text{best}}$, it follows that $R = C_k + C_{\text{best}}$, and the bound for discarding model h_k is therefore given by:

$$\Delta_{h_k,h_{\text{best}}}[t] > (C_k + C_{\text{best}})\sqrt{\frac{\ln(1/\delta)}{2t}}. \tag{4.6}$$

Using this scheme, poorly performing models are discarded from the set of candidates. Since the bound gets tighter as t increases, only one model is eventually maintained on the sensor node.

AMS algorithm

Algorithm 4 shows the pseudocode of the AMS algorithm. It takes as inputs the error tolerance ϵ, the number of candidate models K, the collection of models $\{h_k\}$, and their corresponding model weights $\{C_k\}$[1]. The first model sent to the sink is the one with the lowest model weight.

When the sensor collects a new reading $s[t]$, the AMS runs the function *simulateModel*,

[1]In the particular case where the update rate is used as performance metric, the model weights are all set to 1

Algorithm 4 AMS - Adaptive model selection algorithm.

Input:
ϵ: Error threshold.
K: Number of models.
$\{h_k\}$: Collection of models.
$\{C_k\}$: Model weigths.

Output:
Packets containing model updates sent to the base station (BS).

```
1:  $t \leftarrow 1$
2:  **for** $(k$ in $1 : K)$ **do**
3:      $\theta_k[t] \leftarrow \mathrm{init(h_k)}$
4:      $\theta_k^{\mathrm{last}} \leftarrow \theta_k[t]$
5:      $U_k[t] \leftarrow 1$
6:  **end for**
7:  $k = \arg\min_k C_k$
8:  $h_{\mathrm{best}} \leftarrow h_k$
9:  $\theta_{\mathrm{best}} \leftarrow \theta_k[t]$
10: sendNewModel($h_{\mathrm{best}}, \theta_{\mathrm{best}}$)
11: **while** True **do**
12:     $t \leftarrow t + 1$
13:     $s[t] \leftarrow$ getNewReading()
14:     $\hat{s}[t] \leftarrow$ getPrediction($h, \theta_{\mathrm{best}}$)
15:     **for** $(k$ in $1 : K)$ **do**
16:         simulateModel($k, s[t]$)
17:     **end for**
18:     **if** $(|\hat{s}[t] - s[t]| > \epsilon)$ **then**
19:         $k \leftarrow \arg\min_k U_k[t] * C_k$
20:         $h_{\mathrm{best}} \leftarrow h_k$
21:         $\theta_{\mathrm{best}} \leftarrow \theta_k[t]$
22:         sendNewModel($h_{\mathrm{best}}, \theta_{\mathrm{best}}$)
23:         $\{h_k\} \leftarrow$ racing($\{h_k\}$)            see Equation (4.5)
24:     **end if**
25: **end while**
```

which estimates the weighted update rates $W_k[t]$ for all models h_k. The function is outlined in Algorithm 5. This function first determines whether an update is necessary or not by checking if the current reading estimation $\hat{s}[t] = h_k(x, \theta)$, computed by model h_k at time t, is more than $\pm\epsilon$ off the actual sensor value $s[t]$. The weighted update rate $W_k[t]$ is then updated by the recursive formulation of Equation (4.4). When the parameters of a candidate model can be updated recursively as new sensor readings become available (see Section 4.3), the function *simulateModel* maintains two sets of parameters for each model h_k, namely $\theta_k[t]$ and $\theta_k[t_{last}]$. Parameters $\theta_k[t]$ are continuously updated with incoming data so that the model is constantly refined (e.g., using the recursive least square procedure for autoregressive models, as detailed in section 4.3). On the contrary, as long as no update is necessary for model h_k, parameters $\theta_k[t_{last}]$ remain unchanged since they represent the parameters that would be shared by the sensor node with the base station if h_k was the current model.

Algorithm 5 simulateModel - Algorithm for virtual model updates.

Input:
k: Index of the prediction model to simulate.
$s[t]$: Current measurement.

Output:
h_k is updated.

```
1: $\hat{s}[t] \leftarrow$ getPrediction($h, \theta_k^{\text{last}}$)
2: $\theta_k[t] \leftarrow$ update($\theta_k[t-1], s[t]$)
3: if ($|\hat{s}[t] - s[t]| > \epsilon$) then
4:     $U_k[t] \leftarrow \frac{(t-1)*U_k[t-1]+1}{t}$
5:     $\theta_k^{\text{last}} \leftarrow \theta_k[t]$
6: else
7:     $U_k[t] \leftarrow \frac{(t-1)*U_k[t-1]}{t}$
8: end if
```

After running to completion, the function *simulateModel* returns control to AMS, which then behaves as the standard replicated model approach. It therefore checks whether the absolute value of the difference between the true sensor value $s[t]$ and the prediction $\hat{s}[t] = h_{\text{best}}(x, \theta_{\text{best}})$ does not exceed the tolerated error threshold ϵ. If this threshold is exceeded, the current model h_{best} is assigned the model in $\{h_k\}$ whose $W_k[t]$ is the lowest, and an update of size C_{best} is sent to the sink.

4.3 Model parameter update

The adaptive model selection scheme described in the previous section could be implemented using any collection of prediction models, whether linear or nonlinear. In practice, the two following requirements must however be met. First, the models should comply with the limited computational and memory resources of sensor nodes. Second, it should be generic enough to adapt to different physical realities of the acquired signals.

The class of autoregressive models appears at first glance as a suitable choice for implementing replicated models, as was motivated in previous work on replicated models [87, 79] (cf. Section 3.3). Besides the choice of a class of models, another important design issue consists in defining a procedure to compute the coefficients of the model. For autoregressive models or oder p, the set of coefficients is the vector $\theta = (\theta_1, \theta_2, \ldots, \theta_p)$ used to weight the past measurements in order to get a prediction for the current measurement.

Standard regression

In [87], it was suggested to use a standard regression procedure (cf. Section 2.2.3) based on the last N collected measurements to compute these parameters. More specifically, the coefficients are obtained by computing

$$\theta[t] = (X^T X)^{-1} X^T Y$$

where X is a $(N-p) \times p$ matrix whose i-th row consists in the vector $x_i = (s[t-i], s[t-i-1], \ldots, s[t-i-p])$, $1 \leq i \leq N-p$ and Y is a $(N-p) \times 1$ matrix consisting of the measurements $(s[t], s[t-1], \ldots, s[t-N+p+1])$. In terms of memory, the requirements are in $O(N+p)$. The computational costs are dominated by the computation of the inverse of the $X^T X$ matrix, which is in $O(N^2+p^3)$. The main issue of the standard regression approach is to define a suitable value for N. In particular, too low values of N can lead to poor prediction accuracies, because of the bias/variance tradeoff. This was illustrated in Section 4.1, see Figure 4.2. A high value of N is however costly both in terms of memory and computation.

Adaptive filter

In order to deal with this issue, it was proposed in [79] to compute the autoregression parameters by using the least mean square filter(LMS). The LMS formulation allows to update the parameters every time a new measurement is available, by the following set of equations

$$\begin{aligned}
\hat{s}[t] &= x[t]^T \theta[t] \\
e[t] &= s[t] - \hat{s}[t] \\
\theta[t+1] &= \theta[t] + \mu x[t]^T e[t]
\end{aligned}$$

With LMS, it is no longer necessary to keep a history of the measurements. Besides, the storage and computational requirements are both reduced in $O(p)$. The procedure is therefore much lighter in terms of memory and computational resoures. In terms of prediction accuracy, the efficiency of the procedure however largely depends on a suitable choice for the step size μ, which characterizes the speed of convergence of the filter. In practice, it proves to be difficult to properly define this value.

Recursive least square

We propose to address this issue by using a slightly more computationally demanding procedure called the recursive least square, which provides a recursive way to compute the regression coefficient. The procedure can also be seen as a way to determine the step size adaptively in such a way that the overall mean squared error of the model over time is minimized. The updates of the parameter $\theta[t]$ is achieved by means of the following set of equations

$$
\begin{aligned}
&\hat{s}[t] = x[t]^T\theta[t] \\
&e[t] = s[t] - \hat{s}[t] \\
&L[t] = \frac{P[t]x[t]^T}{1 + x[t]P[t]x[t]^T} \\
&P[t] = P[t] - L[t]x[t]^T P[t] \\
&\theta[t+1] = \theta[t] + L[t]e[t]
\end{aligned}
$$

The matrix $P[t]$ (of size $p \times p$) is an estimate of the inverse correlation matrix $X^T X$ over all time instants up to t. Its initial value $P[0]$ is set to the identity matrix. Compared to the standard regression approach, the RLS therefore allows to use all the past measurements, without storing them on the sensor node. The memory requirements are therefore reduced to an order $O(p^2)$. In terms of computation, the RLS procedure avoids the computation of the inverse of $X^T X$, and therefore reduces the computational requirements to an order in $O(p^2)$.

4.4 Experimental evaluation

Datasets

The experimental evaluation is based on a set of 14 publicly available data sets, collected in real sensor network deployments. The data sets vary in terms of the nature of the observed phenomenon, signal dynamic, sampling frequency and length. They are described in Table 4.1.

The *S Heater* data measures the temperature of a heater as cold water flows, as reported in [81]. The Intel Lab Data from which the set *I Light* is derived, is, to the best of our knowledge, the first large publicly available collection of data derived from a real wireless sensor network deployment and has therefore been used for performing evaluation studies in several works [12, 87, 79]. The data sets *M Temp* and *M Hum* were collected between March 23rd and April 23rd, 2006, by the temperature and humidity sensors of the sensor node 9 in the Montepaldi Farm deployment [29]. All the data sets retrieved from the historical database of the National Data Buoy Center [71] refer to data collected by buoy 41012 during the whole year 2005.

To be able to compare the results obtained from different data sets regardless of the specific physical quantities being examined, the influence of the threshold parameter ϵ is analyzed by considering it as proportional, through a given factor k, to the range r of the signal. The range r was computed by taking the difference between the maximal and minimal values in the time series. The factor k was set to 0.01, 0.05 and 0.2, which implied high, medium, and low bound on the tolerated error. This qualitative notion of high and low tolerance can be illustrated by considering the Montepaldi farm temperature time series for example. For this series, the range of values was $r = 29.8°C$, and therefore the three thresholds considered were $\epsilon = 0.3°C$, $\epsilon = 1.5°C$, and $\epsilon = 6°C$. Figure 4.4 finally illustrates the diversity of the measurement dynamics by reporting as examples the Montepladi farm temperature, the NDBC wind speed, and the Stenman heater temperature time series.

Data set	sensed quantity	sampling period	period	number of samples	source
S Heater	temperature	3 seconds	6h15	3000	[81]
I Light	light	5 minutes	8 days	1584	[18]
M Hum	humidity	10 minutes	30 days	4320	[29]
M Temp	temperature	10 minutes	30 days	4320	[29]
NDBC WD	wind direction	1 hour	1 year	7564	[71]
NDBC WSPD	wind speed	1 hour	1 year	7564	[71]
NDBC DPD	dominant wave period	1 hour	1 year	7562	[71]
NDBC AVP	average wave period	1 hour	1 year	8639	[71]
NDBC BAR	air pressure	1 hour	1 year	8639	[71]
NDBC ATMP	air temperature	1 hour	1 year	8639	[71]
NDBC WTMP	water temperature	1 hour	1 year	8734	[71]
NDBC DEWP	dewpoint temperature	1 hour	1 year	8734	[71]
NDBC GST	gust speed	1 hour	1 year	8710	[71]
NDBC WVHT	wave height	1 hour	1 year	8723	[71]

Table 4.1: Data sets

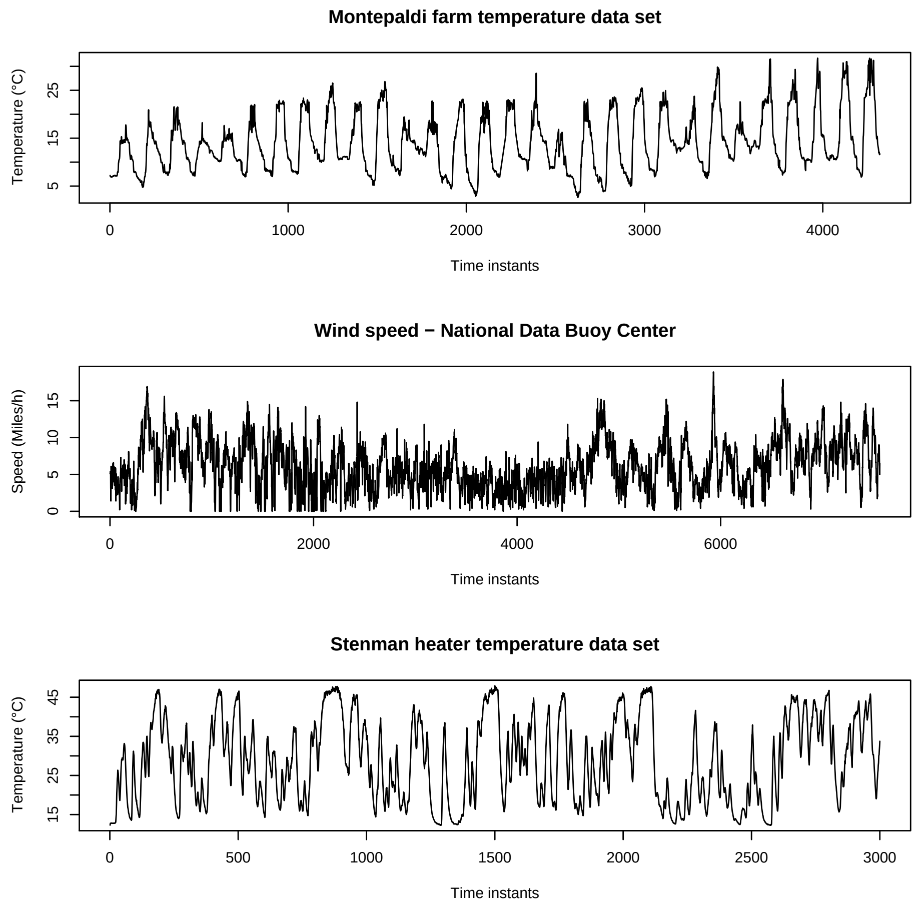


Figure 4.4: Examples of the diversity of the measurement dynamics.

AMS with autoregressive models

We first assess for the fourteen time series the update rates and weighted update rates resulting from the use of a constant model, and of autoregressive models with order ranging from one to five. The accuracy was set at $\epsilon = 0.01r$, where r is the range of the measurements.

According to Equation (4.3), the model weight was set to $C_{\text{AR(p)}} = \frac{24+2p}{24+1}$ for AR(p) models, and to $C_{\text{CM}} = 1$ for the constant model. The update rates and the weighted update rates are reported in the top half and bottom half of Table 4.2, respectively. Bold-faced numbers indicates the model that provided the best performances (Hoeffding test, $\delta = 0.05$). The last column reports the model that was selected by the AMS.

Update rate.							
	CM	AR1	AR2	AR3	AR4	AR5	AMS
S Heater	74	75	61	**59**	**59**	**59**	**AR3**
I Light	**38**	40	39	40	40	39	**CM**
M Hum	53	53	**49**	**50**	**49**	**49**	**AR4**
M Temp	48	48	**45**	**45**	**44**	**44**	**AR4**
NDBC DPD	**65**	85	80	80	80	80	**CM**
NDBC AWP	**72**	**73**	**73**	**73**	**73**	**73**	**CM**
NDBC BAR	51	50	39	39	39	**37**	**AR5**
NDBC ATMP	39	39	**36**	**36**	**36**	**36**	**AR3**
NDBC WTMP	27	27	**21**	**21**	**21**	**20**	**AR5**
NDBC DEWP	57	**52**	**52**	**52**	**52**	**52**	**AR3**
NDBC WSPD	**74**	84	82	83	83	83	**CM**
NDBC WD	85	**81**	**81**	**81**	**81**	**81**	**AR1**
NDBC GST	**80**	81	**80**	**80**	**80**	81	**CM**
NDBC WVHT	58	**56**	**56**	**56**	**56**	**56**	**AR3**
Weighted update rate.							
	CM	AR1	AR2	AR3	AR4	AR5	AMS
S Heater	74	78	**68**	70	76	81	**AR2**
I Light	**38**	42	44	48	51	53	**CM**
M Hum	**53**	55	55	60	62	66	**CM**
M Temp	**48**	50	50	54	56	60	**CM**
NDBC DPD	**65**	89	89	95	102	109	**CM**
NDBC AWP	**72**	75	81	88	93	99	**CM**
NDBC BAR	51	52	**44**	47	49	50	**AR2**
NDBC ATMP	**39**	41	40	43	46	49	**CM**
NDBC WTMP	27	28	**23**	25	27	28	**AR2**
NDBC DEWP	57	**54**	58	62	67	71	**AR1**
NDBC WSPD	**74**	87	92	99	106	113	**CM**
NDBC WD	85	**84**	91	98	104	111	**AR1**
NDBC GST	**80**	84	90	96	103	110	**CM**
NDBC WVHT	**58**	58	63	67	71	76	**CM**

Table 4.2: Update rates (top) and weighted update rates (bottom) obtained using the constant model, and autoregressive models with order up to five. Bold-faced numbers indicate the statistically best performing models.

In all cases, the AMS picked up the model which provided the lowest percentage of updates. When the model weight is not considered, autoregressive models perform better than the constant model for most time series. They however significantly failed to properly model the measurements for the *I Light*(light), the *NDBC DPD* (dominant wave period) and the *NDBC WSPD* (wind speed) time series. These time series are characterized by frequent sudden and sharp changes over time. For this reason, the constant model is better adapted. We note that up to 15% of update savings are gained by using the constant model instead of an autoregressive model on *NDBC DPD*.

In most cases, orders higher than three does not significantly improve the performances in terms of update rates. The only exception is the *NDBC BAR* (barometric pressure) time series. In terms of weighted update rate however, the performances of these models can be strongly impacted by their weight. Using the model weights defined above, autoregressive models of orders higher than three are in all cases significantly worse than other models, and can even lead to percentage of updates higher than 100%, meaning that they caused more communication than the default data collection scheme.

It is also important to note that if the packet overhead was lower than the average size of 24 assumed here, autoregressive models would be even more penalized. With an overhead of 12 bytes for example, the constant model outperforms autoregressive models for all time series. Model weights can therefore strongly change the ability of a complex model to reduce the amount of communication.

The error threshold ϵ is also an important factor in the communication savings which can be obtained. We illustrate this in Figure 4.5, where the weighted update rates are reported for error thresholds $\epsilon = kr$ of the measurement range, with $k \in \{0.01, 0.02, 0.05, 0.1, 0.2\}$. The value r is the range of the measurements for a given time series. The results are reported as boxplots, in which black dots give the distribution of the weighted update rates for each of the fourteen time series.

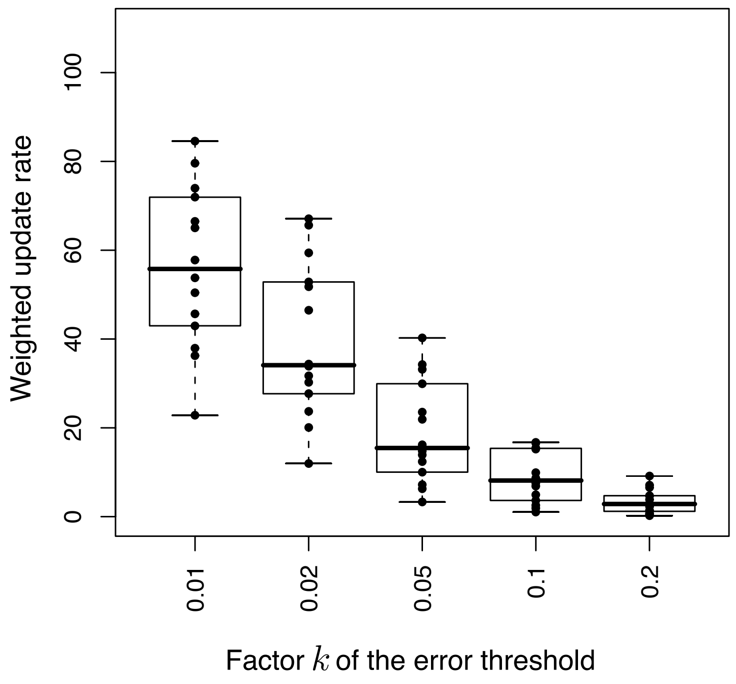


Figure 4.5: Weighted update rate as the error threshold is relaxed.

For a $0.05r$ error threshold, which corresponds to approximations within 5% of the overall measurement range, less than 20% of data are on average sent to the base station. This reduction decreases down to only 5% of updates for an error tolerance of $0.2r$. We should also notice that as the error tolerance increases, the predictive ability of any model usually tends to equal that of the constant model. Therefore, for high error threshold, the AMS selected the constant model.

We finally illustrate in Figure 4.6 the delay taken by the racing algorithm to select the model. The accuracy is set in this example at 0.01, and the packet size at $C_{\mathrm{AR(p)}} = \frac{12+2p}{12+1}$ (minimum packet overhead). The numbers indicate the number of models maintained over time at a sensor node, for each of the fourteen time series. At time $t = 0$, the number of models is six, i.e., the constant model and the five autoregressive models. The racing algorithm was applied from $t = 130$, in order to get a reasonable estimate of the update rate of each model. This explains why subsets of models are discarded at this time instant for most time series.

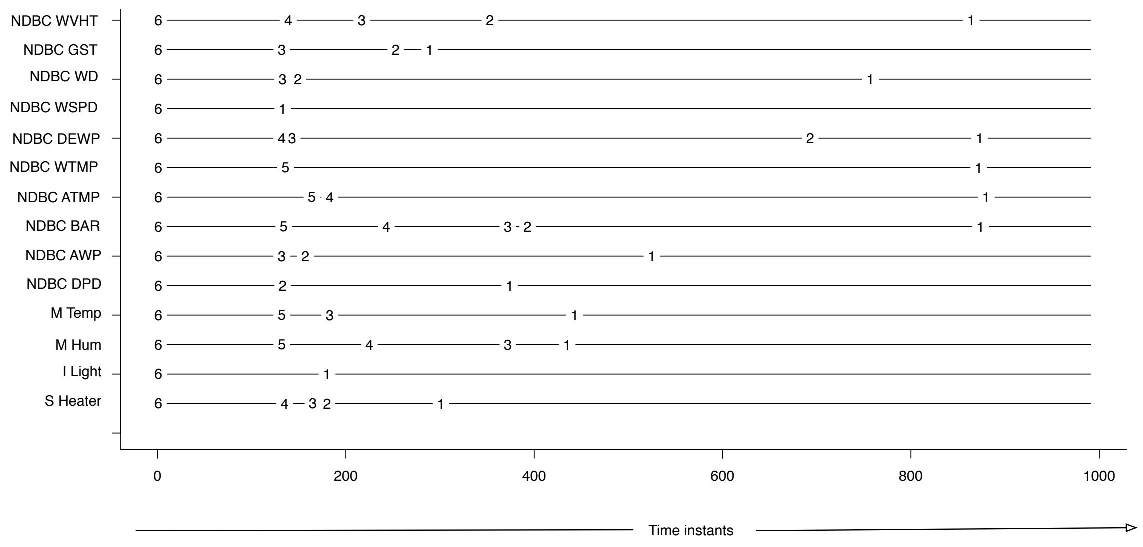


Figure 4.6: Results of the racing algorithm for each of the fourteen time series, with an error tolerance of $0.01r$.

The convergence is reached for all time series in less than 1000 time instants. The convergence may be much quicker. For example, for the I Light (Intel light) temperature measurements, the constant model is deemed as the best one straight after the racing algorithm is called (time instant $t = 130$). The reason is that this time series exhibits rapid and high variations, which cannot be captured by autoregressive models. In most cases however, a few hundred of time instants are required for the racing algorithms to identify the best model in the collection. Finally, it is worth noting that the racing algorithm led in all our experiments to identify the model that provided the best weighted update rate. This reflects the ability of the Hoeffding bound, which is particularly conservative as it does not make any assumption on the error distribution, to properly decide when models can be discarded from the collection of initial models.

4.5 Implementation in TinyOS

This section bridges the gap between the algorithms presented and the real-world. To illustrate how the AMS can be implemented on wireless sensors, we use TinyOS as the programming framework. TinyOS is a free and open source operating system specifically developed for limited resources embedded systems like wireless sensors [54]. It was created in 1999 by the University of Berkeley and Intel, for the WeC wireless sensor platform (Section 2.1.1). It has now been ported to dozen of platforms and numerous sensor boards [85].

Components for scheduling, sensing and transmitting

A TinyOS program is a graph of components. Each component encapsulates a specific set of services, similarly to classes in Oriented Object programming, and interfaces are used to abstract the type of service a component can offer. TinyOS features a large number of interfaces and components, which provide abstractions to sensing, single-hop or multi-hop networking, power management, timers or non-volatile storage for example. The components are written in NesC [26], an extension of the C language designed to define and wire the components together. The components are wired by means of *interfaces*.

The AMS algorithm requires to take measurements periodically, and to transmit data when an update is required. These tasks can be achieved by components that provide the interfaces *Timer*, *ADC* and *SendMsg* (Appendix E), which abstract the functionalities provided by the clock, the sensor hardware and the radio. More precisely,

- The *Timer* interface allows to schedule tasks. It specifies three types of functions, namely `start(char type, uint32_t interval)`, `stop()` and `fired()`, which start, stop or trigger a timer, respectively. The timer interface is used to periodically trigger the AMS algorithm.

- The *ADC* interface is provided by sensor components, and allows to get sensor measurements by means of the functions `getData()` and `dataReady(uint16_t data)`, respectively.

- The *SendMsg* interface is provided by components that implement the transmission of data, such as components which can be used for single or multi-hop routing. The interface specifies two functions, namely `send(uint16_t address, uint8_t length ,TOS_MsgPtr msg)` and `sendDone(TOS_MsgPtr msg, result_t success)`, which allow to send a packet and to ensure that the packet was properly sent.

The AMS components

The NesC specifications distinguishes two types of components, namely the *modules* and the *configurations*. The module provide the implementation (written mostly in C), and may use or provide interfaces. The configurations components create applications by defining how other components are wired.

Let *AMSM* be the module for an implementation of the AMS algorithm. The structure of the file of the *AMSM* module in NesC is

```
module AMSM {
  uses {
    interface Timer;
    interface ADC;
    interface SendMsg;
  }
}
implementation {

 // Implementation in NesC

}
```

The implementation part is written mostly in C, and mainly requires to implement the prediction model algorithms. Let *AMSC* be the configuration component. The role of *AMSC* is to specify the components that implement the interfaces *Timer*, *ADC* and *SendMsg* are implemented, and to form the resulting application. The following configuration

```
configuration AMSC { }
implementation
{
  components AMSM
           , TimerC
           , Temp
           , GenericComm
           ;

  AMSM.Timer -> TimerC.Timer[unique("Timer")];
  AMSM.ADC -> Temp.ADC;
  AMSM.SendMsg -> GenericComm.SendMsg[AM_AMSMSG];
}
```

allows for example to wire the interfaces of the *AMSM* module to the *TimerC*, *Temp* and *GenericComm* components. These components allow the use of the clock, of the temperature sensor, and of single-hop routing [85]. Depending on the application requirements and the sensor hardware, other wiring can be easily defined. The Adaptive Model Selection was implemented in TinyOS by Silvia Santini [78].

4.6 Summary

The AMS algorithm allows to compare different prediction models in a competitive fashion, and aims at selecting the one that minimizes the amount of communication. The selection is carried out thanks to statistical model selection technique called racing. For all series tested in our experimental section, the AMS algorithm correctly identified the prediction model that provided the best performances. The percentage of communication which can be saved depend on the user-defined error-threshold. The higher the threshold, the higher the communication savings. For a reasonably tight error threshold (one hundredth of the sensor signal range), the AMS approach could reduce by 50% on average the amount of data transmitted.

Chapter 5

Distributed Principal Component Analysis

In this chapter, we present a set of contributions aimed at distributing the computation of the principal component analysis (PCA) among the sensor nodes. The PCA is a versatile data processing technique, which finds applications in almost all domains that involve the analysis of multivariate data. In particular, the PCA is widely used for compression, noise filtering, and feature extraction. The technique, presented in Section 2.2.4, is based on linear projections allowing to represent the data in a lower dimensional subspace. The subspace is computed in such a way that most of the variations of the original data are retained. The PCA is a particularly useful technique for sensor networks, where needs for compression, noise filtering, and feature extraction are regularly encountered.

Section 5.1 reviews the potential applications of the technique in the field of sensor networks. Next, we show in 5.2 that linear projections on a lower dimensional subspace can be computed in a distributed manner, by relying on an aggregation service (Section 2.1.4 and 3.4). The proposed approach, called PCAg for *Principal Component Aggregation*, provides a way to model the measurements with varying levels of accuracies, ranging from the average of the measurements over space to the full recovery of the original measurements. The tradeoffs related between the network load and the modeling accuracy are analyzed and quantified in Section 5.3. Then, we investigate in Section 5.4 an approach for identifying the projection subspace in a distributed manner. The proposed algorithm is based on the Power Iteration Method, an iterative technique for computing the eigenvectors of a matrix. In particular we show that this algorithm can compute approximations of the principal component basis under the hypothesis that the sensor measurements collected by distant sensors are uncorrelated. The proposed algorithm is referred to as DCPC, for *Distributed Computation of the Principal Components.* We finally discuss in Section 5.5 the tradeoffs between accuracy and network load caused by this distributed approach.

5.1 Motivations

Our interest in the following will be on the use of the PCA for sensor network measurements. In particular, we will consider the space of dimension $n = S$ of the measurements $s[t] = (s_1[t], s_2[t], \ldots, s_S[t]) \in \mathbb{R}^S$ taken by the sensor nodes at time t.

Thanks to its ability to identify the subspace where information of interest is contained, the use of the PCA for processing the measurements collected by a sensor network has

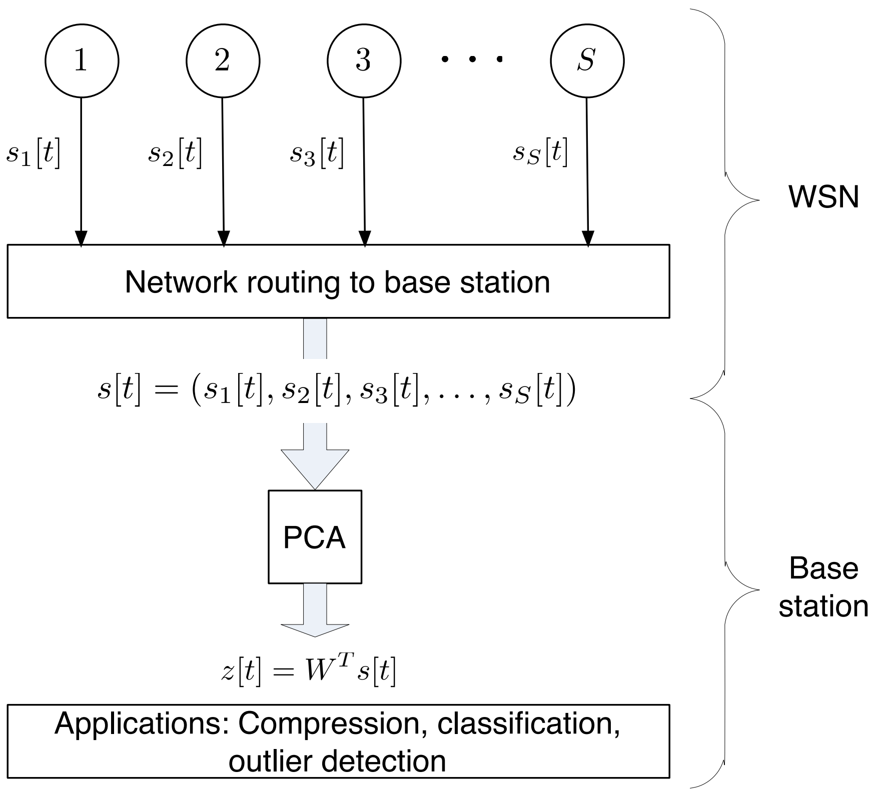

1
2
3
S
$s_1[t]$
$s_2[t]$
$s_3[t]$
$s_S[t]$
Network routing to base station
$s[t] = (s_1[t], s_2[t], s_3[t], \dots, s_S[t])$
PCA
$z[t] = W^T s[t]$
Applications: Compression, classification, outlier detection
WSN
Base station

Figure 5.1: Basic scheme for using the PCA in a sensor network context. Measurements are first collecting from the network at the base station. The PCA is then applied to identify the subspace, and projections $z[t] = W^T s[t]$ can subsequently be computed. Applications include compression, classification, or outlier detection.

been addressed at several occasions [89, 88, 55, 36, 46]. The common approach consists in first collecting at the base station a set of N vectors of measurements $s[t], 1 \leq t \leq N$. These vectors are used to compute the covariance matrix from which the eigenvectors and eigenvalues are computed (cf. Section 2.2.4). The projections on the subspace spanned by the eigenvectors are finally used for applications such as compression, classification, or outlier detection. The PCA is usually applied in a centralized manner at the base station, as illustrated in Figure 5.1.

Compression

The use of the PCA for compression purposes is a classic application of the technique [37, 42]. The PC scores (Section 2.2.4)

$$z[t] = W^T s[t]$$

form the compressed version of $s[t]$, and reduce by a factor $\frac{S}{q}$ the number of elements used for representing the set of measurements. The transform

$$\hat{s}[t] = W z[t]$$

is used to recover an approximation $\hat{s}[t]$ of the original measurements $s[t]$. The compression is lossy, as the projections on the PC basis usually entail some loss that cannot be recovered. This loss is usually quantified in terms of mean squared error (MSE), which is the mean square of the residual error between the original data $s[t]$ and their approximation $\hat{s}[t] = WW^T s[t]$ on the PC basis. Assuming that data are centered for ease of notations, we have

$$MSE(q) = \frac{1}{N} \sum_{t=1}^{N} ||s[t] - WW^T s[t]||^2$$

where the projection matrix W has size $S \times q$. In order to cancel out the effect of the dispersion of the data, the normalized mean squared error defined as

$$NMSE(q) = \frac{\sum_{t=1}^{N} ||s[t] - WW^T s[t]||^2}{\sum_{t=1}^{N} ||s[t]||^2} \tag{5.1}$$

allows to quantify in terms of mean squared error the proportion of loss with respect to the original observations $s[t]$. The MSE and NMSE may be expressed directly by means of the eigenvalues, with

$$MSE(q) = \sum_{k=q+1}^{S} \lambda_k$$

and

$$NMSE(q) = \frac{\sum_{k=q+1}^{S} \lambda_k}{\sum_{k=1}^{S} \lambda_k} = \frac{\sum_{k=1}^{S} \lambda_k - \sum_{k=1}^{q} \lambda_k}{\sum_{k=1}^{S} \lambda_k} = 1 - P(q) \tag{5.2}$$

where λ_k is the eigenvalue associated to the k-th PC, and $P(q) = \frac{\sum_{k=1}^{q} \lambda_k}{\sum_{k=1}^{S} \lambda_k}$ is the proportion of retained variance (see Section 2.2.4). In practice, a common threshold is to set the NMSE so that the 95% of the variations are retained, i.e., $P(q) = 0.05$.

Feature extraction for supervised learning

The PCA is a widely used technique for extracting useful features in order to reduce the number of inputs of a learning problem. It is particularly well-suited for patterns recognition tasks involving a large number of correlated inputs, such as in image recognition tasks [42]. Given the similarity between sensor network data and images, it is easy to come up with a wide range of scenarios where the PCA can improve the accuracy of the learning task.

In the sensor network literature, the use of the PCA as a preprocessing technique for supervised learning problems has for example been motivated for ground vehicle classification with sound sensors [89], posture classification with accelerometer data [62], or inference of parameters of partial differential equations [9].

Outlier detection

Outliers are observations that are far from the rest of the data. For multivariate data, these include observations that are a long way from the rest of the observations in the multidimensional space. A major problem in detecting multivariate outliers is that an observation that is not extreme on any of the original variables can still be an outlier, because it does not conform with the correlation structure of the remainder of the data [42].

Let us assume the presence of an observation $x = (x_1, x_2, x_3) = (2, -2, 0)$ in Figure 2.11. This observation would be very distant from the subspace that approximates the rest of the data. This is because it violates the general pattern of positive correlation between x_1 and x_2. This kind of outliers can be detected by computing the squared error

$$d = ||s[t] - \hat{s}[t]||^2$$

between the observation $s[t]$ and its projection $\hat{s}[t]$ on the PCs. This error is assumed to be low, as the number of PCs is chosen so that this squared error is minimized. The use of a threshold on this error can be used to determine if the observation is an outlier. The approach has been applied for example to the detection of network traffic anomalies in [36], and to vibration sensor fault detection in [44].

5.2 Principal component aggregation

This section shows that an aggregation service (Sections 2.1.4 and 3.4), can be used to compute the principal component scores within the network. The approach, called *principal component aggregation* (PCAg), relies on the ability of an aggregation service to compute scalar products in a distributed manner. We then show that the approach can be used to provide either unbounded or bounded approximation errors.

Implementation in an aggregation service

Let $z[t] = W^T s[t]$ be the projection of $s[t]$ on the PC basis at time t, and let us assume that the basis has size q, i.e., $z[t] \in \mathbb{R}^q$. The main result here is that the computation of the k-th coordinate $z_k[t]$ can be performed within the network by means of an aggregation service if each node i has available the i-th entry $w_{(i,k)}$ of W. Indeed, the scalar product is a sum of multiplications

$$z_k[t] = \sum_{i=1}^{S} w_{(i,k)} s_i[t]$$

which can be implemented in an aggregation service by means of the following set of primitives

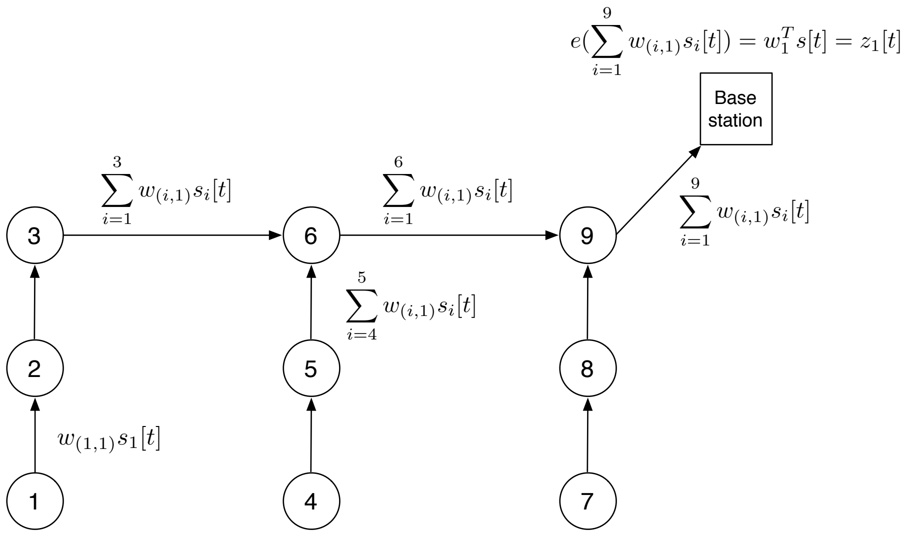
$$\begin{cases} init(s_i[t]) &= \langle w_{(i,k)} s_i[t] \rangle = \langle X \rangle \\ f(\langle X \rangle, \langle Y \rangle) &= \langle X + Y \rangle \\ e(\langle X \rangle) &= X \end{cases} \tag{5.3}$$

The partial state record is a scalar that represents a part of the scalar product $w_k^T s[t]$. Once all sensors have brought their contribution $w_{(i,k)} s_i[t]$, the evaluator returns the complete scalar product $\sum_{i=1}^{S} w_{(i,k)} s_i[t]$ which gives the coordinate $z_k[t]$ of the set of measurements $s[t]$ on the k-th principal component. The aggregation process is illustrated in Figure 5.2.

The aggregation can be extended to several, or all coordinates. The partial state record then takes the form of a vector, whose size is the number q of coordinates computed by means of aggregation.

Initialization

For the computation of q PC scores, the q elements $w_{(i,k)}$, $1 \leq k \leq q$, of the matrix W (forming its i-th row) are assumed to be available at each sensor i. A distributed approach (the



$e(\sum_{i=1}^{9} w_{(i,1)}s_i[t]) = w_1^T s[t] = z_1[t]$
Base station
$\sum_{i=1}^{3} w_{(i,1)}s_i[t]$
$\sum_{i=1}^{6} w_{(i,1)}s_i[t]$
$\sum_{i=1}^{9} w_{(i,1)}s_i[t]$
$\sum_{i=4}^{5} w_{(i,1)}s_i[t]$
$w_{(1,1)}s_1[t]$
3
6
9
2
5
8
1
4
7

Figure 5.2: Aggregation service at work for computing the projections of the set of measurements on the first basis vector.

DCPC) will be developed lated in Section 5.4 For now, we assume the following centralized scenario:

- A set of N vectors of measurements $s[t]$, $1 \leq t \leq N$, is first collected at the base station.
- The sample covariance matrix $\hat{\Sigma}$ of these vectors, and the eigenvectors of $\hat{\Sigma}$ are computed at the base station using the N vectors of measurements.
- The elements $w_{(i,k)}$ of these eigenvectors are communicated to each sensor i.

Spatio-temporal transformation

The extension of the principal component aggregation scheme to the temporal domain can be achieved by means of Multichannel Single Spectrum Analysis (MSSA) [42]. The technique consists in integrating lagged measurements $s[t-\delta]$, $0 \leq \delta \leq \tau - 1$, with $\tau \geq 1$, in the computation of the covariance matrix. Let

$$\tilde{s}_i[t] = (s_i[t], s_i[t-1], \ldots, s_i[t-\tau+1])$$

be a vector that contains the past measurements up to time $t-\tau+1$ for sensor i, and let

$$x[t] = (\tilde{s}_1[t], \tilde{s}_2[t], \ldots, \tilde{s}_S[t])$$

be the vector composed of the set of last τ measurements for the S sensors. The size of this measurement vector is τS, and the sample covariance matrix $\hat{\Sigma}$ of these measurements over time has size $\tau S \times \tau S$. The computation of the q first principal components leads to a W matrix of size $\tau S \times q$ which can be used to aggregate the measurements in the spatiotemporal domain

$$z[t] = W^T x[t] = \left(\begin{array}{l} \sum_{i=1}^{\tau S} w_{(i,1)} x_i[t] \\ \dots \\ \sum_{i=1}^{\tau S} w_{(i,q)} x_i[t] \end{array} \right).$$

Note now that $1 \leq q \leq \tau S$. In-network aggregation can be performed exactly as with spatial compression, except each projection requires a sensor node to perform τ aggregations. More specifically, each sensor i is initialized with the set of elements $\{w_{(i',k)}, w_{(i'+1,k)}, \dots, w_{(i'+\tau-1,k)}\}$ for each of the k PC, where $i' = (i-1) * \tau + 1$ points to the first element in a PC that is related to sensor i. The primitives are as follows :

$$\left\{ \begin{array}{rcl} init(s_i[t], s_i[t-1], \dots, s_i[t-\tau+1]) & = & \langle \sum_{\delta=0}^{\tau-1} w_{(i'+\delta,k)} s_i[t-\delta] \rangle = \langle X \rangle \\ f(\langle X \rangle, \langle Y \rangle) & = & \langle X + Y \rangle \\ e(\langle X \rangle) & = & X \end{array} \right. \tag{5.4}$$

The initializer is the only primitive which changes, and its inputs are now the set of elements $\{s_i[t], s_i[t-1], \dots s_i[t-\tau+1]\}$. The use of lagged measurements requires the nodes to keep in memory the set of the last τ measurements. In the rest of this thesis, we will for the sake of simplicity assume that $\tau = 1$, although all computations can be easily extended to the temporal domain.

Unbounded and bounded approximations

The PCAg provides a way to perform a *lossy compression* of the data generated by the sensors. The loss may be estimated by the amount of variance not captured by the PCs. The coordinates obtained at the base station may be used to get unbounded or bounded approximations, where the bound is a user-defined error threshold ϵ.

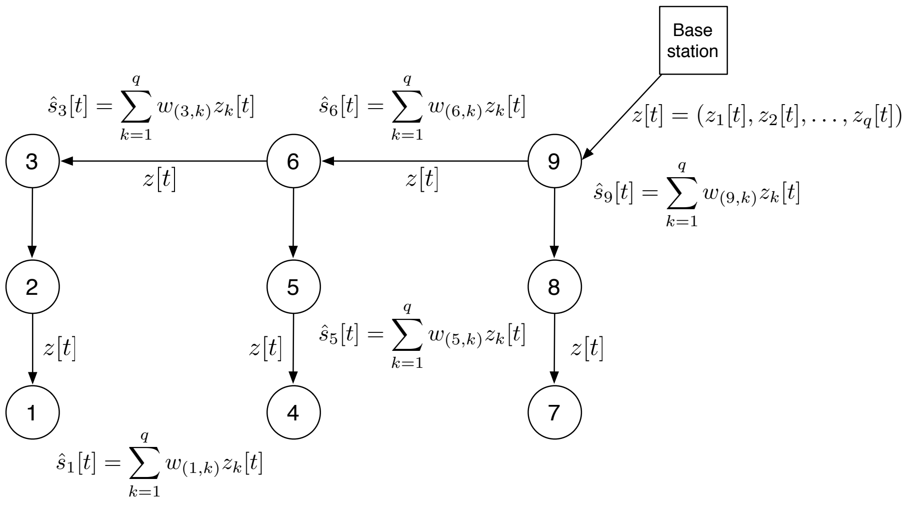
Unbounded approximations: Given a set of q coordinates $z[t] = (z_1[t], \dots, z_q[t])$ computed by the aggregation service, the approximation $\hat{s}[t]$ of the original $s[t]$ is obtained at the base station by transforming the vector of coordinates $z[t]$ back to the original basis by the evaluator function

$$\begin{array}{rcl} e(z_1[t], \dots, z_q[t]) & = & (\hat{s}_1[t], \dots, \hat{s}_S[t]) \\ & = & W z[t] \end{array}$$

which returns an approximation $\hat{s}[t]$ of $s[t]$ by using the q principal components. Note that if $q = S$, the evaluation step returns the exact vector of measurement $s[t]$. Otherwise, if the number of coordinates q is less than S, the evaluation returns an optimal approximation to the observation $s[t]$ in the mean square sense.

Bounded approximations: In the PCAg scheme, the approximations obtained at the base station cannot be compared to the true observations obtained by the sensors. Indeed, the approximation $\hat{s}_i[t]$ of the i-th $(1 \leq i \leq S)$ sensor observation at time t is given by:

$$\hat{s}_i[t] = \sum_{k=1}^{q} z_k[t] w_{(i,k)}$$



Base station
$z[t] = (z_1[t], z_2[t], \ldots, z_q[t])$
$\hat{s}_3[t] = \sum_{k=1}^{q} w_{(3,k)} z_k[t]$
$\hat{s}_6[t] = \sum_{k=1}^{q} w_{(6,k)} z_k[t]$
$\hat{s}_9[t] = \sum_{k=1}^{q} w_{(9,k)} z_k[t]$
$\hat{s}_5[t] = \sum_{k=1}^{q} w_{(5,k)} z_k[t]$
$\hat{s}_1[t] = \sum_{k=1}^{q} w_{(1,k)} z_k[t]$
$z[t]$
1
2
3
4
5
6
7
8
9

Figure 5.3: Feedback of the coordinates $z[t]$ in the network. Each sensor i can compute locally the approximation $\hat{s}_i[t]$, and compare it with its true measurement $s_i[t]$.

The terms $\{w_{(i,k)}\}$ are available at each node. It is therefore enough to send the vector $z[t]$ in the network for each sensor to be able to compute locally the approximation retrieved at the base station. The strategy is illustrated in Figure 5.3.

Sensors can then send a notification to the base station when the approximation error is greater than some user defined threshold ϵ. This scheme, called BPCAg for bounded approximation principal component aggregation, guarantees that all data eventually obtained at the base station are within $\pm\epsilon$ of their actual measurements.

5.3 PCAg accuracy and network load

In this section, we analyze the tradeoffs of the PCAg between data accuracy and network load. The most important parameter is q, the number of principal components to retain, since it governs the relationships between the different tradeoffs.

Accuracy

In terms of accuracy, the parameter q is directly related to the amount of information that is retrieved from the network. This amount is quantified in terms of proportion of retained variance, by the relation

$$P(q) = \frac{\sum_{k=1}^{q} \lambda_k}{\sum_{k=1}^{S} \lambda_k} \tag{5.5}$$

as detailed in Section 2.2.4. The function $P(q)$ increases monotonically with q, since increasing the number of principal components necessarily increases the amount of retained variance. The rate of increase of $P(q)$ depends on the variances of the measurements taken by each sensor, and of the correlations existing between these measurements. Let us denote by σ_i the variance of the measurements collected by sensor i, and consider the three following cases.

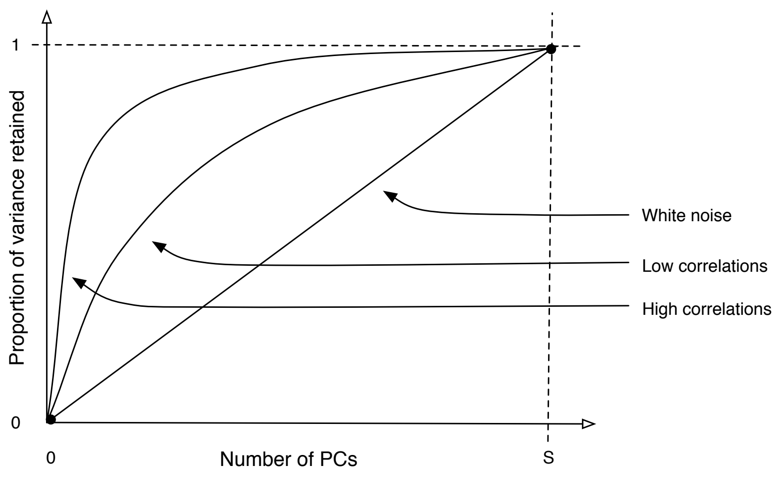


Figure 5.4: Qualitative representation of the proportion of retained variance as a function of the number of PCs. The higher the correlations in the data, the higher the proportion of retained variance for a fixed number of PCs.

First, when the measurements are uncorrelated and have the same variance $\sigma = \sigma_1 = \sigma_2 = \ldots = \sigma_S$ (white noise), eigenvalues all equal σ. The PCA is in this case of no use, since no vector of $\mathbb{R}^S$ can be used to better represent the data. The function P increases linearly with q, with

$$P(q) = \frac{q}{S}.$$

Second, if the measurements are uncorrelated but have different variances, the set of eigenvalues equals $\{\lambda_k\} = \{\sigma_i\}$, with $k, i \in \{1, \ldots, S\}$. The PCA leads in this case to select the sensor nodes whose measurements have the highest variance, and

$$P(q) = \frac{\sum_{k \in \mathcal{S}_q} \sigma_k}{\sum_{k=1}^{S} \sigma_k}$$

where $\mathcal{S}_q$ is the subset of size q that contains the sensor nodes i whose variances σ_i are the highest.

Finally, when there exists correlation between sensor measurements, the PCA allows to find a new basis that combines the measurements in such a way that the proportion of retained variance on this basis is higher that the proportion retained by selecting a subset of sensors. The proportion is quantified by Equation (5.5) and satisfy

$$P(q) = \frac{\sum_{k=1}^{q} \lambda_k}{\sum_{k=1}^{S} \lambda_k} \geq \frac{\sum_{k \in \mathcal{S}_q} \sigma_k}{\sum_{k=1}^{S} \sigma_k}.$$

The number of PCs to compute depends on the approximation accuracy required by the observer, and is application dependent. In practice, determining the value q can be achieved by first collecting a set of N measurements at the base station. The PCA is then applied, and a validation procedure is carried out to assess the performances of the applications as a function of the number of PCs. The validation procedure may be, for example, a K-fold cross validation. This techniques will be used in the experimental evaluation in Chapter 6.

Network load

The PCAg falls in the set of learning approaches based on aggregation, such as the distributed regression presented in Section 3.4. As a result of aggregation, the computation of q PCs imply q transmissions for all sensors, no matter their depth in the routing tree. We push further the analysis made in Section 3.4 by explicitly considering the network loads L_i (Section 3.1) related to both transmissions and receptions

$$L_i = \mathrm{Rx}_i + \mathrm{Tx}_i.$$

For the sake of our analysis, we more particularly focus on the network loads resulting from three types of *actions*, referred to as D, A and F actions.

- *D action*: A D action, where D stands for *default*, consists in retrieving the measurements from the whole set of S sensors at the base station. D actions are the basis of the default data collection strategy, where all measurements are collected at the base station without in-network processing.
- *A action*: An A action consists in retrieving an aggregate of size one, by means of an aggregation service.
- *F action*: An F action consists in feeding back to all sensors an aggregated value obtained by means of an A action.

The following analysis compares the network load related to the D, A and F actions, respectively. We consider the number of packets processed by each node (i.e., number of receptions and transmissions) in an ideal case where overhearing, collisions or retransmissions are ignored. A routing tree is assumed to connect all sensor nodes to the base station. The set of children of a node i in the tree is denoted $\mathcal{C}_i$, and their number is denoted $|\mathcal{C}_i|$. $\mathcal{S}_i$ denotes the set of sensors in the subtree whose root is the i-th sensor, and $|\mathcal{S}_i|$ denotes the size of this subtree. In the tree topology represented in Figure 5.5, we have for example $\mathcal{C}_6 = \{3, 5\}$ with $|\mathcal{C}_6| = 2$, and $\mathcal{S}_6 = \{1, 2, 3, 4, 5\}$ with $|\mathcal{S}_6| = 5$.

During a D action, all the measurements are routed to the base station by means of the routing tree. The load is the lowest at leaf nodes, which only send one packet per epoch, while the load is the highest at the root node which processes $2S-1$ packets ($S-1$ receptions and S transmissions) per epoch. The network load at the i-th sensor node depends on the routing tree, and amounts to

$$L_i^{\mathrm{D}} = 2|\mathcal{S}_i| - 1 \tag{5.6}$$

packets per epoch.

During an A action, each node only sends one packet and receives a number of packets which depends on its number of children. The total number of packets processed is therefore

$$L_i^{\mathrm{A}} = |\mathcal{C}_i| + 1 \tag{5.7}$$

per epoch. The load is the lowest at leaf nodes, which only have 1 packets to send, while the load is the highest at the node whose number of children is the highest.

Finally, an F action generates a network load of two packets for all non-leaf nodes (one reception and one transmission for forwarding the packet to the children) and of one packet for the leaves (one reception only), i.e.,

$$L_i^{\mathrm{F}} = 1 + 1(|\mathcal{C}_i| \neq 0) \tag{5.8}$$

where 1(.) is the indicator function.

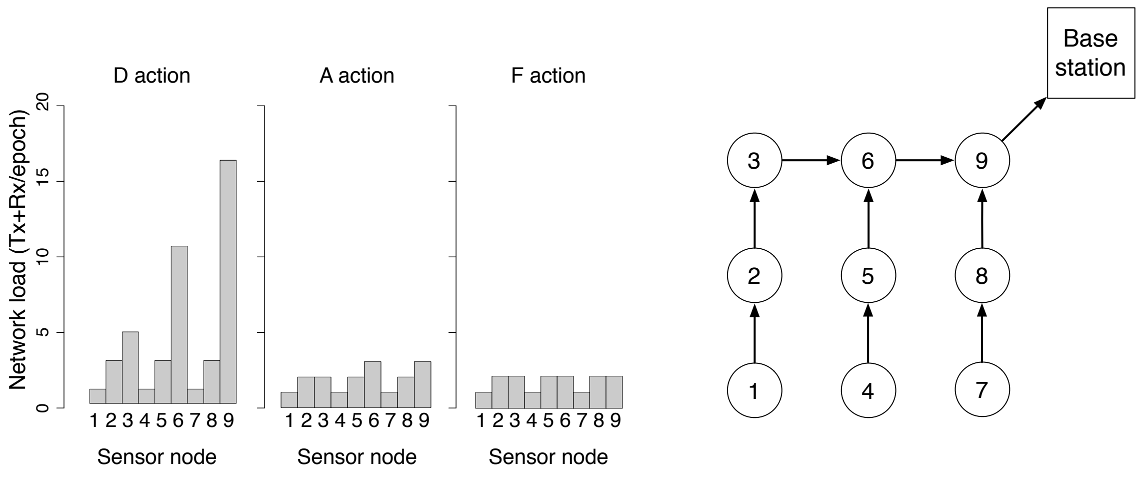


Figure 5.5: Per-node load for the D, A and F actions in a grid routing topology.

Figure 5.5 gives an example of the network load in terms of number of receptions and transmissions for each of the three actions, in a grid network of nine sensors. The load is better balanced in A and F actions than in D actions. For example, an A action causes a load of three packets for nodes 6 and 9 (two receptions and one transmission), whereas the load is only of 1 for leaf nodes 1, 4, and 7 (one transmission only).

The highest network load is much higher for a D action than for A and F actions. Table 5.1 summarizes the highest network load caused by each of the three actions, where $i_{\mathcal{C}}^* = \arg\max_i C_i$ is the node whose number of children is the highest.

Action type	Highest network load
D action	$L_{\max}^{\mathrm{D}} = 2S - 1$
A action	$L_{\max}^{\mathrm{A}} = \lvert C_{i_{\mathcal{C}}^*}\rvert + 1$
F action	$L_{\max}^{\mathrm{F}} = 2$

Table 5.1: Summary of the highest network loads entailed by D, A and F actions.

The definition of D and A actions allows us to study under which conditions the PCAg reduce the load in comparison to the default data collection scheme. In particular, concerning the highest network load (HNL), it is easy to equate $L_{\max}^{\mathrm{D}}$ and $L_{\max}^{\mathrm{A}}$ to get

$$\begin{aligned} L_{\max}^{\mathrm{PCAg}} < L_{\max}^{\mathrm{D}} &\Leftrightarrow qL_{\max}^{\mathrm{A}} < L_{\max}^{\mathrm{D}} \\ &\Leftrightarrow q(|C_{i_{\mathcal{C}}^*}| + 1) < 2S - 1 \\ &\Leftrightarrow q < \frac{2S-1}{|C_{i_{\mathcal{C}}^*}| + 1} \end{aligned} \tag{5.9}$$

where $L_{\max}^{\text{PCAg}}$ is HNL of the principal component aggregation for computing $q \leq S$ PC scores. The load related to the number of packet receptions in the network load metric implies that the ability of the PCAg to reduce the HNL is actually topology dependent. Different topologies will therefore be considered in our experimental results for studying the efficiency of the PCAg.

In the BPCAg, the q PC scores must be communicated back to all sensor nodes so that they can locally compute the approximation $\hat{s}_i[t]$ obtained at the base station. Depending on the number of sensors whose true measurement is more than ϵ away from the approximation, S additional transmissions may be required. More precisely, for a node i, the network load is the sum of

- qL_i^{A} packets as q A actions are performed for aggregating the PC scores,
- qL_i^{F} packets as q F actions are performed for communicating back the PC scores to the sensor nodes, and
- up to $2|\mathcal{S}_i| - 1$ packets if all sensor nodes in $\mathcal{S}_i$ send an update.

Therefore, denoting by L_i^{BPCAg} the network load sustained by sensor i in the BPCAg

$$qL_i^{\text{A}} + qL_i^{\text{F}} \leq L_i^{\text{BPCAg}} \leq qL_i^{\text{A}} + qL_i^{\text{F}} + 2|\mathcal{S}_i| - 1$$

Given that $L_i^{\text{A}} = |\mathcal{C}_i| + 1$ and $L_i^{\text{F}} = 1 + 1(|\mathcal{C}_i| \neq 0)$, this more precisely gives

$$q(|\mathcal{C}_i| + 2) \leq L_i^{\text{BPCAg}} \leq q(|\mathcal{C}_i| + 3) + 2|\mathcal{S}_i| - 1 \tag{5.10}$$

The highest network load caused by the bounded approximation PCAg is sustained by the node that maximizes the upper bound in (5.10), and we have

$$L_{\max}^{\text{BPCAg}} = \max_i q(|\mathcal{C}_i| + 3) + 2|\mathcal{S}_i| - 1. \tag{5.11}$$

This upper bound is the worst case, in which all approximations obtained by the PC scores are more than ϵ away from the true measurements. In fact, the error tolerance ϵ plays an important role in the ability of the BPCAg to reduce the network load, and determines whether L_i^{BPCAg} is closer to $q(|\mathcal{C}_i| + 2)$ or $q(|\mathcal{C}_i| + 3) + 2|\mathcal{S}_i| - 1$ in Equation (5.10).

Two extreme are $\epsilon = 0$ and $\epsilon = r$, where r is the range of the measurements. In the former case, the exact measurements are required, and therefore the HNL is indeed given by Equation (5.11). The best strategy in this case is to use the default data collection. This load is represented in Figure 5.6 as the top horizontal dashed line. In the case of $\epsilon = r$, all measurements are bound to be within $\pm\epsilon$ of the true measurements as the mean is considered as the exact approximation. The highest network load therefore is in $O(q)$, as in the unbounded error PCAg, and is represented by the diagonal solid line in Figure 5.6.

In between, the loads implied depend on both the number of PCs collected and the error tolerance ϵ. Given an error tolerance, a too low number of PCs is likely to entail many approximation errors, and therefore many updates from the sensor nodes. Increasing the number of PCs allows to improve the approximations and thus to reduce the number of updates. At the same time however, the load is increased as more PC scores are aggregated. The minimum load is attained for a number of PCs that depends on the error tolerance. For tight error tolerance, a higher number of PCs is likely to be required, whereas for large

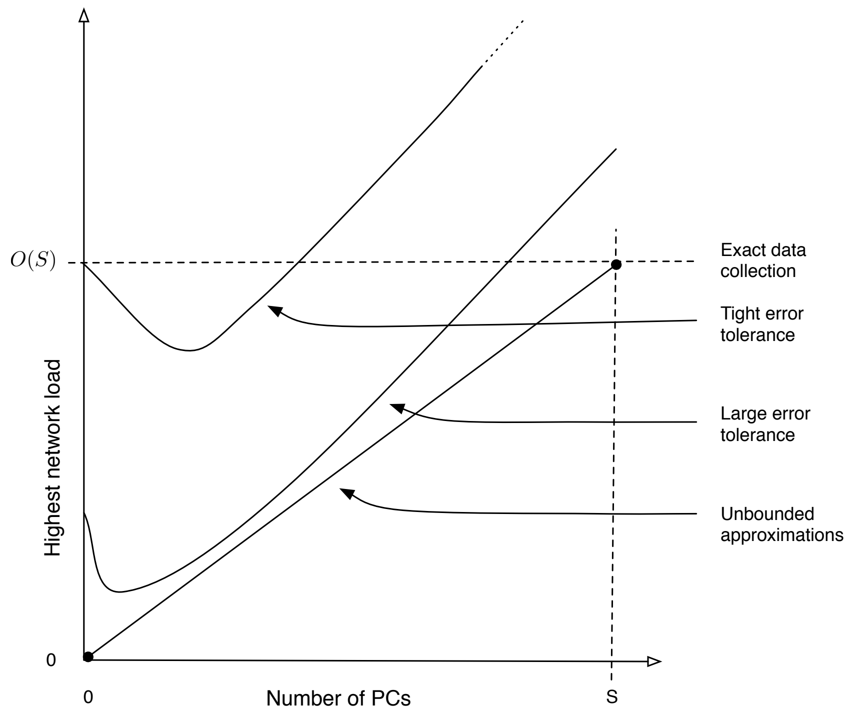


Figure 5.6: Qualitative relationship between the highest network load and the number of PCs in bounded approximations. The minimum load depends both on the accuracy required and the number of PC scores computed.

error tolerance, a lower number of PCs is likely to lead to the minimum load. Figure 5.6 qualitatively illustrates this tradeoff.

Summary

The ability of the PCAg to reduce the network load mainly depends on the correlations existing in the measurements. The higher the correlations, the higher the savings in terms of highest network load. An important characteristic of the PCAg is to imply an equal number of transmissions for all nodes, and therefore the better balance the network load among sensor nodes. In particular, it provides a way to significantly reduce the load of the root node in comparison to exact data collection. The network loads caused by the bounded error PCAg are more difficult to assess, given that an additional number of transmissions, up to S, may be needed to ensure the approximation accuracy required by the observer. Exact data collection is hence likely to perform better when the error tolerance is low.

5.4 Distributed computation of the principal components

The PCAg requires an initialization procedure whereby each node i is initialized with the elements $w_{(i,1)}, \ldots, w_{(i,q)}$ of the principal components $w_1, \ldots, w_q$. We first describe in more details the centralized procedure which was outlined in Section 5.2. Next, we investigate an alternative approach, where this initialization process is to a certain extent distributed by means of an aggregation service. The approach proposed relies on the power iteration method (PIM), a classic iterative technique for computing the eigenvectors of a matrix [5], and on a simplifying assumption on the covariance structure which allows to estimate the covariance matrix in a distributed way. Finally, we compare the communication, computational, and memory costs of the different approaches.

Centralized approach

The centralized approach works by first collecting a set of N vectors of sensor measurements $s[t] \in \mathbb{R}^S$, $t \in \{1, \dots, N\}$ at the base station. These measurements are stored in a $N \times S$ matrix $X[N]$, whose t-th row-vector is the vector $s[t] \in \mathbb{R}^S$.

Once the measurements have been collected, the sample covariance matrix $\hat{\Sigma}[N]$ of the measurements $s[t]$, $1 \leq t \leq N$, is obtained by computing

$$\begin{aligned} \hat{\Sigma}[N] &= \frac{1}{N-1} \sum_{t=1}^{N} (s[t] - \hat{\mu}[N])(s[t] - \hat{\mu}[N])^T \\ &= \frac{1}{N-1} X[N]^T X[N] - \frac{N}{N-1} \hat{\mu}[N] \hat{\mu}[N]^T \end{aligned} \tag{5.12}$$

where $\hat{\mu}[N] = \frac{1}{N} \sum_{t=1}^{N} s[t]$ is an estimate of the average vector of observations $s[t]$, $1 \leq t \leq N$,. The covariance matrix may alternatively be computed in a recursive manner as new vectors of observations are made available at the base station. Denoting by $\hat{\sigma}_{(i,j)}[N]$, $1 \leq i, j \leq S$, the elements of $\hat{\Sigma}[N]$ at time $t = N$, it follows from Equation (5.12) that

$$\hat{\sigma}_{(i,j)}[N] = \frac{1}{N-1} r_{(i,j)}[N] - \frac{1}{N(N-1)} r_i[N] r_j[N] \tag{5.13}$$

where

$$\begin{aligned} r_i[N] &= \sum_{t=1}^{N} s_i[t] = r_i[N-1] + s_i[N] \\ r_{(i,j)}[N] &= \sum_{t=1}^{N} s_i[t] s_j[t] = r_{(i,j)}[N-1] + s_i[N] s_j[N]. \end{aligned}$$

The details of these derivations are given in Appendix B. Once the covariance matrix is estimated, the principal components can be computed using a standard eigendecomposition method [5] and its elements communicated to the sensor nodes.

Highest network load: The centralized estimation of the covariance matrix from a set of N vectors of observations first requires the collection of the measurements from all sensors during N epochs. This is done using the default data collection, and requires N D actions as defined in Section 5.3. Given that $L_{\max}^{\mathrm{D}} = 2S - 1$, the highest network load for the centralized estimation of the covariance matrix is

$$L_{\max}^{\mathrm{Cov_{cent}}} = N(2S - 1). \tag{5.14}$$

Once the eigenvector decomposition is performed at the base station, the base station transmits the estimates principal components to the sensor nodes. This requires to send q vectors of size S, and therefore requires qS F actions. Given that $L_{\max}^{\mathrm{F}} = 2$, the highest load is

$$L_{\max}^{\mathrm{EV_{cent}}} = 2qS. \tag{5.15}$$

Computational and memory costs: From Equation (5.13), the computational cost related to the covariance matrix at the base station is $O(NS^2)$. The memory cost for storing the matrix of observations and the covariances is $O(NS + S^2)$ or $O(S^2)$ if recursive updates are relied on. As far as the estimation of the principal components is concerned, the cost of standard eigendecomposition algorithm is $O(S^3)$ in terms of computation and $O(S^2)$ in terms of memory [5].

Distributed approach

The computation of the eigenvectors and eigenvalues of a symmetric matrix is a numerical algebra problem for which a range of different techniques, mainly based on iterative algebraic transforms, have been designed [28]. Among them, we identified that the power iteration method (PIM), a well-known approach to the eigendecomposition problem, could be applied in a distributed manner under the assumption that the covariance matrix is sparse. More precisely, the sparsity must be such that the covariances between the sensor nodes that cannot directly communicate is zero. This assumption, which we call *local covariance hypothesis*, tough strong is supported by the fact that for a number of realistic scenarios, the spatial correlations between sensor measurements decrease with the distance separating the sensors.

In the following, we first describe the PIM, and then detail how it can be implemented in an aggregation service. Next, we investigate the behavior of the method when the local covariance hypothesis is not satisfied. Finally, we provide an analysis of communication, memory, and computational costs of the method.

Power Iteration Method

Let $\hat{\Sigma}$ be an estimate of the covariance matrix. The power iteration method is initialized with a random vector $v[0]$, which serves as a (random) initial guess for the estimate of the principal eigenvector of $\hat{\Sigma}$. At the t-th iteration, the vector $v[t+1]$ is obtained by multiplying $\hat{\Sigma}$ by $v[t]$, and by normalizing it. It can be shown (see Appendix C) that $v[t]$ converges to the principal eigenvector $\hat{w}_1$ of $\hat{\Sigma}$, under the condition that $v[0]$ is not strictly orthogonal to the principal eigenvector, and that principal eigenvectors are unique (no multiplicity in the eigenvalues). The convergence rate is exponential in the ratio of the two principal eigenvalues. The convergence criteria can be defined either as a minimum variation δ for $v[t]$, or as the highest number of iterations $t_{\max}$ [5, 28]. Algorithm 6 outlines the different steps.

Note that, as the method converges to the principal eigenvector $\hat{w}_1$, the normalizing factor $\|v[t]\|$ converges to the associated eigenvalue $\hat{\lambda}_1$, as by definition:

$$\hat{\Sigma}\hat{w}_1 = \hat{\lambda}_1\hat{w}_1. \tag{5.16}$$

In practical settings, the power method quickly converges to a linear combination of eigenvectors whose eigenvalues are close, or to the eigenvector whose eigenvalue is the highest if eigenvalues are well separated (Appendix C). As our purpose here is to find the subspace that minimizes the approximation error, eigenvectors with close eigenvalues can be thought

Algorithm 6 PIMmain - Power iteration method for computing the main eigenvector and eigenvalue

Input:
$\hat{\Sigma}$: Estimate of the sample covariance matrix.
$t_{\max}$: Maximum number of iterations.
δ: Minimum increment for $v[t]$.

Output:
Estimated eigenvector $\hat{w}_1$ and eigenvalue $\hat{\lambda}_1$.

```
1: $v[0] \leftarrow$ random initialization such that $||v[0]|| = 1$
2: $t \leftarrow 0$
3: repeat
4:     $v'[t] \leftarrow \hat{\Sigma} v[t]$
5:     $v[t+1] \leftarrow \frac{v'[t]}{||v'[t]||}$
6:     $t \leftarrow t+1$
7: until $t > t_{\max}$ and/or $\|v[t+1] - v[t]\| \leq \delta$
8: return $v[t]$ as $\hat{w}_1$ and $\frac{||v[t+1]||}{||v'[t]||}$ as $\hat{\lambda}_1$
```

of as generating a subspace where similar amount of information are retained along any vector of the subspace. The convergence to a linear combination of eigenvectors with close eigenvalues is therefore deemed acceptable as far as principal component aggregation is concerned. The outcome of Algorithm 6 provides estimates $\hat{w}_1$ and $\hat{\lambda}_1$ of the principal eigenvector and its eigenvalue, respectively.

Computation of subsequent eigenvectors

The standard way to employ the power iteration method in order to find the other eigenvectors (up to the number q) is the *deflation* method which consists in applying the PIM to the covariance matrix from which we have removed the contributions of the k principal eigenvectors already computed [5]. The vector $v[t]$ is first orthogonalized with respect to the previously estimated eigenvectors:

$$\begin{aligned} v''[t] &= (\hat{\Sigma} - \sum_{l=1}^{k} \hat{w}_l \hat{\lambda}_l \hat{w}_l^T) v[t] \\ &= \hat{\Sigma} v[t] - \sum_{l=1}^{k} \hat{w}_l \hat{\lambda}_l \hat{w}_l^T v[t] \end{aligned} \tag{5.17}$$

for all eigenvectors $\hat{w}_l$ and eigenvalues $\hat{\lambda}_l$, $1 \leq l \leq k$. The resulting vector $v'[t]$ is then normalized to obtain a unit vector

$$v[t+1] = \frac{v''[t]}{||v''[t]||}. \tag{5.18}$$

The resulting process is given by Algorithm 7.

Algorithm 7 PIMdeflation - Power iteration algorithm with deflation

Input:
$\hat{\Sigma}$: Estimate of the sample covariance matrix.
$t_{\max}$: Maximum number of iterations.
δ: Minimum increment for $v[t]$.
q: Number of pairs of eigenvectors and eigenvalues to estimate.

Output:
q pairs of estimated eigenvector $\hat{w}_k$ and eigenvalue $\hat{\lambda}_k$, $1 \leq k \leq q$.

```
1: $k \leftarrow 0$
2: **repeat**
3:     $k \leftarrow k+1$
4:     $t \leftarrow 0$
5:     $v[0] \leftarrow$ random initialization such that $||v[0]|| = 1$
6:     **repeat**
7:         $v'[t] \leftarrow \hat{\Sigma} v[t]$
8:         $v''[t] \leftarrow v'[t] - \sum_{l=1}^{k-1} \hat{\lambda}_l (v'[t]^T \hat{w}_l)\hat{w}_l$
9:         $v[t+1] \leftarrow \frac{v''[t]}{||v''[t]||}$
10:        $t \leftarrow t+1$
11:    **until** $t > t_{\max}$ or $||v[t+1] - v[t]|| \leq \delta$
12:    $\hat{\lambda}_k \leftarrow \frac{||v'[t]||}{||v[t]||}$
13:    $\hat{w}_k \leftarrow v[t+1]$
14: **until** $k = q$
15: return $\{\hat{w}_k, \hat{\lambda}_k\}$, $1 \leq k \leq q$
```

Distributed implementation

We detail in the following how the different steps of Algorithm 7 can be achieved in a distributed manner. An important parameter of the procedure is the radio communication range of the sensor nodes, which causes a tradeoff between the accuracy of the eigendecomposition and the communication costs incurred. More specifically, the proposed procedure requires to assume that entries $\sigma_{(i,j)}$ of the covariance matrix Σ are zero if sensor nodes i and j cannot directly communicate with one another. This assumption, called the *local covariance hypothesis*, entails an approximation of the covariance matrix. A special case arises when all sensors can communicate with one another, in which case the accuracy of the eigendecomposition is as good as the centralized version. As the number of sensor nodes not in communication range increases, this accuracy is likely to decrease. The set of sensor nodes with which a node i can directly communicate is denoted $\mathcal{N}_i$, and is called the neighborhood of sensor node i. In the case where all sensor nodes can communicate directly with one another, we have $\mathcal{N}_i = \mathcal{S} \setminus i$. We also assume that in a neighborhood, radio links are symmetric, i.e., that if sensor node i receives data from node j, then sensor node j receives data from node i:

$$j \in \mathcal{N}_i \Rightarrow i \in \mathcal{N}_j.$$

The summary of the different steps of the procedure are given in Algorithm 8, whose structure is the same as Algorithm 7 for clarity. The details of the main steps are given below.

- **Inputs**: Before running the power iteration method, each sensor has locally computed estimates of the covariances between its measurements and the measurements of sensor nodes in its neighborhood $\mathcal{N}_i$. These estimates can be obtained by an initialization stage where each sensor node i broadcasts its measurement at each epoch, and receives the measurements collected by the set of sensors $j \in \mathcal{N}_i$. Using the recursive formulation of Equation (5.13), each sensor i can update over time the quantities r_j and $r_{(i,j)}$ for $j \in \mathcal{N}_i$, and get after N epochs estimates of the covariances $\hat{\sigma}_{(i,j)}$ between its measurements and the measurements of the sensor nodes of its neighborhood. The $\hat{\sigma}_{(i,j)}$ of $j \notin \mathcal{N}_i$ are assumed to be null (local covariance hypothesis). The parameters $t_{\max}$, δ and q are provided by the observer.

Algorithm 8 PIM_DPCA - Distributed implementation of the power iteration method

Input:
$\hat{\sigma}_{(i,j)}, \forall j \in \mathcal{N}_i$: Each sensor i locally has estimates of the covariances between its measurements and those of its neighbors $j \in \mathcal{N}_i$.
$t_{\max}$: Maximum number of iterations.
δ: Minimum increment for $v[t]$.
q: Number of pairs of eigenvectors and eigenvalues to estimate.

Output:
$\{\hat{w}_k, \hat{\lambda}_k\}$, $1 \leq k \leq q$ are distributed in the network.

```
1: $k \leftarrow 0$
2: **repeat**
3:     $k \leftarrow k + 1$
4:     $t \leftarrow 0$
5:     $\forall i$ node $i$ is initialized with a $v_i[0] = \frac{1}{\sqrt{S}}$
6:     **repeat**
7:         Nodes exchange locally their $v_i[t]$, and compute $v'[t] = \hat{\Sigma} v[t]$ in parallel
8:         $\{(v'[t]^T \hat{w}_l)\}_{1 \leq l \leq k-1}$ are computed by the aggregation service, $v''[t]$ is updated
9:         $\|v''[t]\|$ is computed by the aggregation service, $v[t+1]$ is updated
10:        $t \leftarrow t + 1$
11:    **until** $t > t_{\max}$ or $\|v[t+1] - v[t]\| \leq \delta$
12:    $\hat{\lambda}_k \leftarrow \frac{\|v'[t]\|}{\|v[t]\|}$
13:    $\hat{w}_k \leftarrow v[t+1]$
14: **until** $k = q$
```

- **Step 5: Initialization of** $v[0]$: A convenient initialization is to set $v_i[0] = \frac{1}{\sqrt{S}}$, so that $||v[0]|| = 1$.

- **Step 7: Computation of** $v'[t] = \hat{\Sigma} v[t]$: This computation, which corresponds to the distributed version of step 7 in Algorithm 7, is performed in parallel. This means that each sensor i only computes the i-th element $v'_i[t]$

$$v'_i[t] = \sum_{j=1}^{S} \hat{\sigma}_{(i,j)} v_j[t]$$

of the vector $\hat{\Sigma} v[t]$. As we made the assumption that

$$\forall i \in \{1, \dots, S\} \ , \ \forall j \notin \mathcal{N}_i \ , \ \hat{\sigma}_{(i,j)} = 0$$

the sum can be simplified

$$v'_i[t] = \sum_{j \in \mathcal{N}_i} \hat{\sigma}_{(i,j)} v_j[t].$$

In order to compute this sum, each node i broadcasts at each epoch its own $v_i[t]$, and receives the $v_j[t]$ of the sensor nodes in its neighborhood $\mathcal{N}_i$.

- **Step 8: Orthogonalization** $v''[t] \leftarrow v'[t] - \sum_{l=1}^{k-1} (v'[t]^T \hat{w}_l) \hat{w}_l$: Let us first consider the scalar products $\{v'[t]^T \hat{w}_l\}_{1 \leq l \leq k-1}$

  $$\sum_{i=1}^{S} v'_i[t] \hat{w}_{(i,l)}$$

  with $1 \leq l \leq k-1$. These products can be computed by the aggregation service, by means of the following primitives

  $$\begin{cases} init(v'_i[t]) &= \langle (v'_i[t] \hat{w}_{(i,1)}, v'_i[t] \hat{w}_{(i,2)}, \dots, v'_i[t] \hat{w}_{(i,k-1)}) \rangle \\ f(\langle X \rangle, \langle Y \rangle) &= \langle X + Y \rangle \\ e(\langle X \rangle) &= X. \end{cases}$$

  The resulting $k-1$ scalar products $\{v'[t]^T \hat{w}_l\}_{1 \leq l \leq k-1}$ are communicated back from the base station to all the sensors. Each sensor node i can then locally update the i-th element

  $$v''_i[t] \leftarrow v'_i[t] - \sum_{l=1}^{k-1} \hat{\lambda}_l (v'[t]^T \hat{w}_l) \hat{w}_{(i,l)}$$

  of $v''[t]$ (cf. Equation (5.17)) as it locally has the elements $v'_i[t]$, $\hat{\lambda}_l$ and $\hat{w}_{(i,l)}$.

- **Step 9: Orthonormalization** $v[t+1] = \frac{v''[t]}{||v''[t]||}$: The computation of the norm $||v''[t]||$ can be done by an aggregation service using the following primitives:

  $$\begin{cases} init(v''_i[t]) &= \langle (v''_i[t]^2) \rangle \\ f(\langle X \rangle, \langle Y \rangle) &= \langle X + Y \rangle \\ e(\langle X \rangle) &= \sqrt{X}. \end{cases}$$

  The norm is then communicated back from the base station to all sensor nodes, which can locally compute the i-th element $v_i[t+1] = \frac{v''_i[t]}{||v''[t]||}$.

These different steps are synthesized graphically in Figure 5.7.

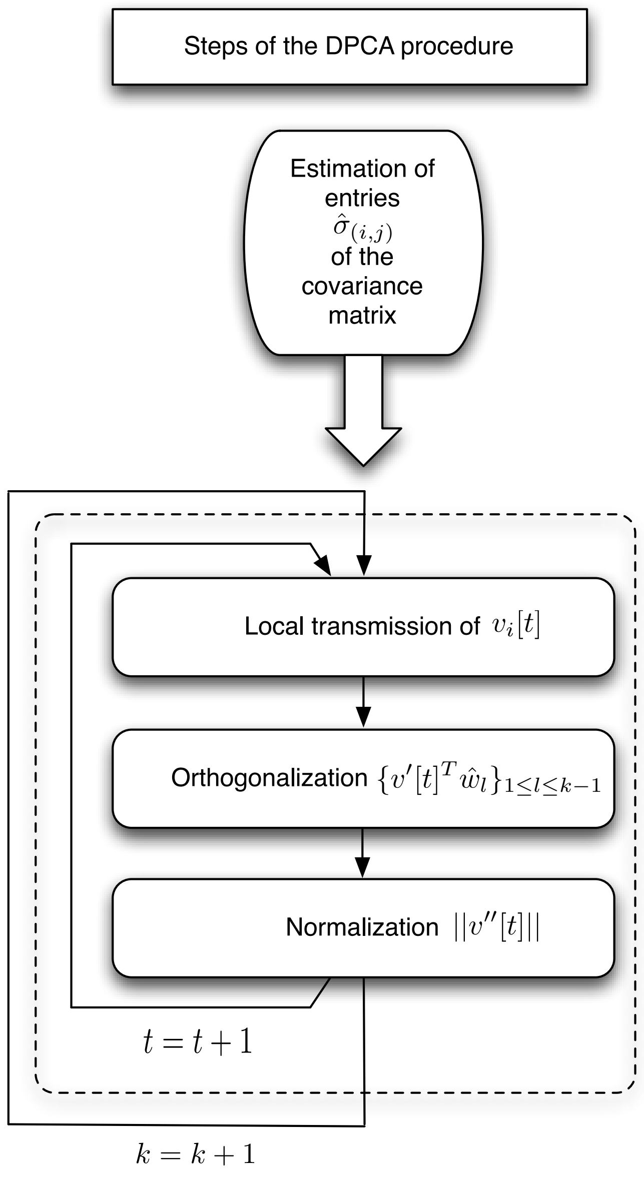

Steps of the DPCA procedure
Estimation of entries $\hat{\sigma}_{(i,j)}$ of the covariance matrix
Local transmission of $v_i[t]$
Orthogonalization $\{v'[t]^T \hat{w}_l\}_{1 \le l \le k-1}$
Normalization $||v''[t]||$
$t = t + 1$
$k = k + 1$

Figure 5.7: Summary of the different steps of the DPCA procedure.

Communication and computational costs

We analyze here the costs related to the computation of the covariance matrix and its distributed eigendecomposition.

Highest network load:

- For the covariance matrix, each update of the covariances $\hat{\sigma}_{(i,j)}$, $j \in \mathcal{N}_i$, requires a node i to send one packet (its measurement) and to receive $|\mathcal{N}_i|$ packets (its neighbors' measurements). Assuming that N updates are applied to get an estimate of the covariance matrix, the network load sustained by a node i amounts to

$$L_i^{\mathrm{Cov_{dist}}} = N|\mathcal{N}_i|. \tag{5.19}$$

Let $i^*_{\mathcal{N}}$ be the node that has the largest number of neighbors, i.e., $i^*_{\mathcal{N}} = \arg\max_i |\mathcal{N}_i|$. The highest network load is therefore

$$L_{\max}^{\mathrm{Cov_{dist}}} = N|\mathcal{N}_{i^*_{\mathcal{N}}}|. \tag{5.20}$$

- For the power iteration method, the number of packets processed by a node i during an iteration is the sum of the packets processed during the following stages
  - Measurement broadcast : One transmission and $|\mathcal{N}_i|$ receptions.
  - Normalization stage: It implies one action of type A and one action of type F.
  - Orthogonalization stage: It implies $k-1$ actions of type A and F respectively, where k is the index of the principal component computed.

  Let $t_{\mathrm{conv}}(k)$ be the number of iteration for the convergence of the computation of the k-th eigenvector. The network load sustained by node during for the computation of q components is therefore

$$\begin{aligned} L_i^{\mathrm{EV_{dist}}} &= \sum_{k=1}^{q} t_{\mathrm{conv}}(k)(&& \textit{Sum for all PCs from 1 to q} \\ & \quad 1+|\mathcal{N}_i| && \textit{Measurement broadcast} \\ & \quad +L_i^A + L_i^F && \textit{Normalization} \\ & \quad +(k-1)(L_i^A + L_i^F)) && \textit{Orthogonalization} \\ &= \sum_{k=1}^{q} t_{\mathrm{conv}}(k)(1+|\mathcal{N}_i| + k(L_i^A + L_i^F)). \end{aligned}$$

  Given that $L_i^A = |\mathcal{C}_i| + 1$ and $L_i^F = 1 + 1(|\mathcal{C}_i| \neq 0)$, the network load can be expressed as

$$L_i^{\mathrm{EV_{dist}}} = \sum_{k=1}^{q} t_{\mathrm{conv}}(k)(1+|\mathcal{N}_i| + k(|\mathcal{C}_i| + 2 + 1(|\mathcal{C}_i| \neq 0)). \tag{5.21}$$

  The highest network load for the computation of the q first principal components amounts to

$$L_{\max}^{\mathrm{EV_{dist}}} = \max_i \sum_{k=1}^{q} t_{\mathrm{conv}}(k)(1+|\mathcal{N}_i| + k(|\mathcal{C}_i| + 2 + 1(|\mathcal{C}_i| \neq 0)). \tag{5.22}$$

Computational and memory costs: For the computation of covariances, the updates of r_j and $r_{(i,j)}$ demand a number of operations proportional to the neighborhood size $|\mathcal{N}_i|$ (cf. Equation (5.13)). Let $i^*_{\mathcal{N}_i} = \arg\max_i |\mathcal{N}_i|$. The highest computational cost therefore is in $O(N|\mathcal{N}_{i^*_{\mathcal{N}}}|)$. In terms of memory requirement, it is in $O(|\mathcal{N}_{i^*_{\mathcal{N}}}|)$, i.e., the number of covariance values that the node locally computes.

Regarding the power iteration method, the cost of the computation of $\hat{\Sigma}v[t]$ is in $O(|\mathcal{N}_i|)$ for the sensor node i. The cost of the orthogonalization step is in $O(k|\mathcal{C}_i|)$ and the cost of the normalization step is in $O(1)$. The overall highest computational cost therefore amounts to $O(t_{\max}(q^2|\mathcal{C}_{i^*_{\mathcal{C}}}|+q|\mathcal{N}_{i^*_{\mathcal{N}}}|))$. Regarding memory costs, each node i needs to maintain variables for storing its local $w_{(i,k)}$ and its neighbors parameters v_j, $j \in \mathcal{N}_i$. The complexity of the highest memory cost is therefore $O(q+|\mathcal{N}_{i^*_{\mathcal{N}}}|)$.

5.5 DCPC accuracy and network load

The distributed computation of the principal components (DCPC) aims at reducing the communication and computational costs of the centralized eigendecomposition approach. In most cases, the approach however entails a loss of accuracy in the computation of the eigenvectors and eigenvalues. The following provides an analysis of the tradeoffs implied by the distributed approach.

Accuracy

A loss of accuracy is likely to occur for the following reasons. First, the local covariance hypothesis entails in most cases an approximation of the covariance matrix. Second, the power iteration method is less exact than more exact eigendecomposition method such as QR method [42, 28], particularly when the number of eigenvectors to compute becomes large.

Let $\hat{\Sigma}$ be the centralized estimate of the sample covariance matrix, and $\hat{\Sigma}'$ be the estimate obtained using the local covariance hypothesis. The centralized estimate is computed on the basis of N vectors of measurements $s[t]$, $1 \leq t \leq N$. It is assumed that the number of collected measurements is sufficient to provide a good approximation of the covariance matrix. The covariance structure of the measurements is furthermore assumed to be stable over time. The PCAg is not adapted to scenarios where these conditions cannot be met.

If the local covariance hypothesis is satisfied, the estimate $\hat{\Sigma}'$ obtained by the distributed procedure equals the centralized estimate $\hat{\Sigma}$. In the general case however, the hypothesis is not satisfied. $\hat{\Sigma}'$ hence becomes an approximation of $\hat{\Sigma}$, where all entries $\hat{\sigma}_{(i,j)}$ for which $j \notin \mathcal{N}_i$ are set to zero.

Pseudo covariance matrix: Failure to satisfy the local covariance hypothesis can lead to an approximation $\hat{\Sigma}'$ that does not have the geometric structure of a covariance matrix. Such matrices are called *pseudo covariance matrix* [77]. Consider for example the case of three sensor nodes i, j and k whose measurements have unit variance and are perfectly correlated. Let us further assume that i can communicate with j and k, but that j and k cannot communicate with one another, as illustrated in Figure 5.8. Perfect correlation of i, j and i and k implies that j and k are also perfectly correlated. By setting the entry $\hat{\sigma}'_{(j,k)}$ to zero, the matrices $\hat{\Sigma}$ does not represent a valid covariance structure anymore.

It can be shown that a sufficient and necessary condition for a matrix to be a covariance matrix is that all its eigenvalues are positive or null. Such matrices are called positive semi definite (PSD). The issue of a non PSD estimate of a covariance matrix commonly arises when the covariance matrix is estimated from data with missing measurements for example. The problem is extensively discussed in [77], where different approaches are reviewed to transform the estimated matrix in such a way that it becomes PSD. These approaches however work in a centralized manner.

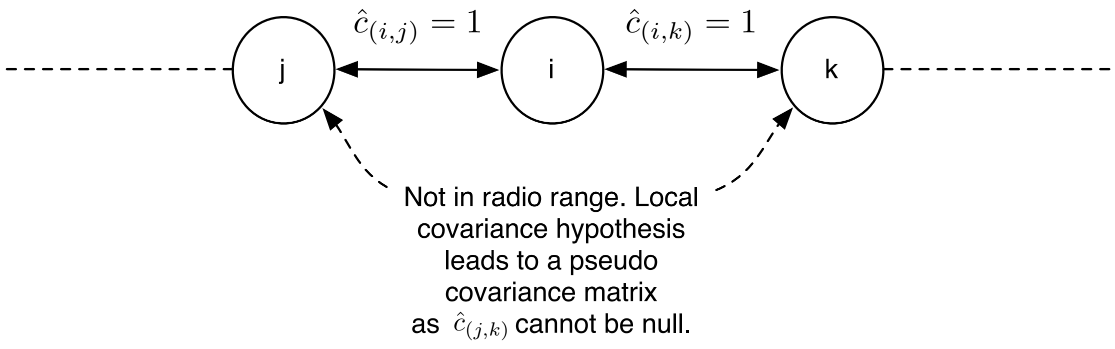

$\hat{c}_{(i,j)} = 1$
$\hat{c}_{(i,k)} = 1$
j
i
k
Not in radio range. Local covariance hypothesis leads to a pseudo covariance matrix as $\hat{c}_{(j,k)}$ cannot be null.

Figure 5.8: The local covariance hypothesis can lead a group sensor nodes with strongly correlated measurements to give a pseudo covariance matrix if some of the nodes are not in radio range.

We identify that one of these approaches, called the eigenvalue approach, could however be adapted to the DCPC. The rationale of the eigenvalue approach consists in computing the eigendecomposition of $\hat{\Sigma}'$. Denoting by w' and λ' the eigenvectors and eigenvalues of $\hat{\Sigma}'$, we have

$$\hat{\Sigma}' = \sum_{k=1}^{S} \lambda'_k w'_k w'^T_k. \tag{5.23}$$

The transform consists in removing the contributions with negative eigenvalues from the sum in (5.23), which gives by definition a PSD matrix.

A property of an eigenvector w_k with a negative eigenvalue λ_k is that the sign of all the elements Σw_k is the opposite of the sign of the elements of w_k, as by definition $\Sigma w_k = \lambda_k w_k$. In the power iteration method, a test can be added after the convergence at step 12 in Algorithm 8 to check whether the eigenvalue is negative. The test could consist in simply comparing the sign of the value of the first element of $v[t]$ to that of $v[t+1]$. In order to make the test more robust, we suggest averaging this comparison over all elements by computing

$$\mathrm{sign}(\sum_{\mathrm{i}=1}^{\mathrm{S}} \mathrm{sign}(\mathrm{v_i[t]v_i[t+1]})) \tag{5.24}$$

where sign(x) is a function

$$\begin{aligned} \mathrm{sign} : \mathbb{R} &\rightarrow \{-1, 0, 1\} \\ x &\mapsto \begin{cases} -1 & \text{if x} < 0 \\ 0 & \text{if x} = 0 \\ 1 & \text{if x} > 0. \end{cases} \end{aligned}$$

The DCPC procedure is therefore stopped if Equation (5.24) returns -1, as all the remaining eigenvalues are assumed to be null. This way, the transform of a non PSD estimate $\hat{\Sigma}'$ in a PSD matrix is done via the power iteration method itself.

Impact on the principal components accuracy: The use of the power iteration method inevitably leads to approximation errors in the computation of the eigenvectors and eigenvalues. The accuracy depends on the criterion δ and $t_{\max}$ chosen to stop the iterative sequence in Algorithm 6. The speed of convergence of the algorithm depends on the ratio of the two main eigenvalues, and is quick when these are well separated (see Appendix C). In practice, from our experiments, a maximum of 10 iterations were observed to provide good approximations.

In cases where eigenvalues are close, the convergence speed can be much slower. This issue is however not critical, as the purpose of the DCPC is not to find the *exact* eigenvectors, but to find a basis of vector able to represent the collected measurements. Close eigenvalues indicate that the corresponding eigenvectors have a similar ability to represent the measurements. In such cases, the convergence time is slow to reach the eigenvector whose eigenvalue is truly the highest, but the PIM however quickly converges to a linear combination of the eigenvectors whose eigenvalues are among the highest. This means that, even if the vector obtained by the PIM before convergence is not close to the true main eigenvector, the vector obtained is effective for approximating the measurements.

The approximation of the covariance matrix resulting from the local covariance hypothesis leads to more serious discrepancies between the true and estimated eigenvalues and eigenvectors. In particular, using an approximated covariance matrix that does not reflect the true covariance structure of the sensor measurements inevitably leads to the identification of a vector basis that does not optimally capture the correlations. As of now, we can only provide some intuition about the fact that the vectors obtained by the DCPC are a *best guess*, given the lack of information entailed by the local covariance hypothesis. This intuition is supported by the following arguments.

Information on some of the covariances is still better than no information at all. The basis computed by the PIM is suboptimal, but consistent with the information available about the covariance structure. In particular, it may lead to use several vectors to represent a subspace that could have been representing by only one vector. This is for example the case if a group of sensor nodes have highly correlated measurements, but that the communication links prevent one subset to communicate with another subset of sensors in this group. Two basis vectors are therefore required to represent the measurements, whereas just one would have been enough if all the sensors in the group were all able to directly communicate.

Such a neat separation is however unlikely in practice. A more realistic situation is that some correlated sensors cannot communicate with one another, but that the correlation existing between their measurements appear indirectly in the incomplete covariance matrix. More precisely, as in the example of Figure 5.8, the assumption that $\sigma_{(j,k)}$ is zero makes the incomplete covariance matrix not PSD. An interesting point is that the transform, using the early stopping of the PIM as described above by ignoring eigenvectors with negative eigenvalues, actually makes it PSD, in such a way that an amount of covariance is assumed between these two sensors. Further theoretical work is needed to support these arguments. We could only test the validity of the approach by means of simulations, and the results obtained are encouraging. This will be the subject of Chapter 6.

Network load

Table 5.2 summarizes the highest network loads entailed by the centralized and distributed approach for the computation of the covariance matrix and its eigendecomposition (Equations (5.14), (5.15), (5.20) and (5.22)). In order to better put into evidence the tradeoffs between the different approaches, the orders of magnitude of these HNLs are also reported.

	Details of the highest Network load	Order of magnitude
Centralized		
Covariance	$L_{\text{max}}^{\text{Cov}_{\text{cent}}} = N(2S-1)$	$\sim O(NS)$
Eigenvectors	$L_{\text{max}}^{\text{EV}_{\text{cent}}} = 2qS$	$\sim O(qS)$
Distributed		
Covariance	$L_{\text{max}}^{\text{Cov}_{\text{dist}}} = N\lvert\mathcal{N}_{i^*_{\mathcal{N}}}\rvert$	$\sim O(N\lvert\mathcal{N}_{i^*_{\mathcal{N}}}\rvert)$
Eigenvectors	$L_{\text{max}}^{\text{EV}_{\text{dist}}} = \max_i \sum_{k=1}^{q} t_{\text{conv}}(k)(1+\lvert\mathcal{N}_i\rvert$ $+k(\lvert\mathcal{C}_i\rvert+2+1(\lvert\mathcal{C}_i\rvert \neq 0))$	$\sim O(t_{\max}(q^2\lvert\mathcal{C}_{i^*_{\mathcal{C}}}\rvert + q\lvert\mathcal{N}_{i^*_{\mathcal{N}}}\rvert))$

Table 5.2: Summary of the highest network loads entailed by the centralized and distributed approach for the computation of the covariance matrix and its eigendecomposition.

The main advantage of the distributed approach is that the HNLs do not depend on the number of sensor nodes S. The distributed computation of the covariance matrix can lead to significant communication savings, particularly if the network size is large. If the local covariance hypothesis is satisfied, the distributed approach reduces the HNL by a factor $\frac{S}{|\mathcal{N}_{i^*_{\mathcal{N}}}|}$ with no loss in accuracy. However, as discussed above, the hypothesis is rarely satisfied in practice, and therefore the gains in HNL come at the cost of an approximation of the sample covariance matrix.

The distributed computation of the eigenvectors causes a network load that increases quadratically with the number of PCs to compute. This quadratic increase is due to the orthogonalization stage, which implies that the computation of the k-th PC requires the transmission of $k-1$ aggregates containing the scalar products of the k-th PC with the $k-1$ PCs previously computed. As a result, the distributed approach can reduce the network load for the computation of a low number of PCs, but is not appropriate for computing a large number of PCs. The maximum number of iteration $t_{\max}$ emphasizes the relationship between the accuracy of the distributed approach and the network load incurred. More precisely, the power iteration method provides a way to get approximations of the eigenvectors, and the accuracy of these approximations depends mainly on the number of iterations that are carried out. Each iteration however involves some communication, and there is therefore a tradeoff between accuracy and network load. In all our experiments, 10 iterations were enough for providing good approximations, as will be detailed in Chapter 6.

Finally, the HNLs entailed by the distributed approaches depend on two network dependent quantities, namely the largest neighborhood size $|\mathcal{N}_{i^*_{\mathcal{N}}}|$ and the largest number of children $|\mathcal{C}_{i^*_{\mathcal{C}}}|$ in the routing tree. It is worth mentioning that some networking techniques can be used to modify these quantities. These will be addressed in our future work in Chapter 7.

5.6 Summary

The PCA provides a powerful way to compress data, and to extract information from sensor network data. This chapter first described a distributed approach called PCAg which computes the principal component scores along a routing tree. The procedure provides varying compression accuracies, and balances in a more even manner the network loads among sensors. The use of this approach requires an initialization stage, for which a distributed procedure was investigated. This procedure is based on the hypothesis that distant sensors have uncorrelated measurements. It can lead to communication savings when the number of PCs required is low, but can lead to accuracy loss if the local covariance hypothesis is not satisfied.

Chapter 6

Experimental Evaluation of the Distributed Principal Component Analysis

In this chapter, we present a set of experimental results which illustrate the tradeoffs between accuracy and network load with the DPCA. We rely on two data sets for our experiments. The first one consists of artificial measurements which simulate the appearance and disappearance of three different spatial patterns in a monitored environment. Its use is motivated by the fact that (i) it illustrates the possible use of DPCA for approximation and classification applications, and (ii) it makes the interpretation of the results easier as the dynamic of the measurements is known. The second data set consists in real world temperature measurements, obtained from a network of 52 sensors which was deployed at the Intel Laboratory of Berkeley in 2004 [39]. This data set is used to validate, on real-world data, the results obtained on the artificial data set.

The first section of this chapter details how the network is simulated in order to assess the network loads entailed by the DPCA. The next two sections report the results for the artificial and the real-world data sets, respectively. The same structure is used in both sections, and consists of 5 parts:

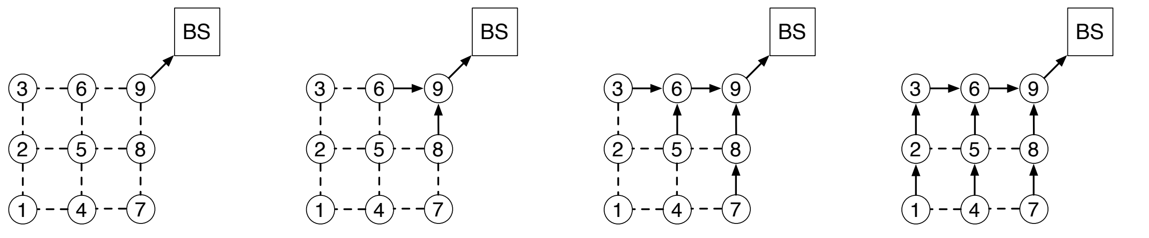
- First, we assess the ability of the PCA to properly represent the measurements with a small number of components. This part only focuses on the data, and is independent of network considerations. The full covariance matrix is considered.
- Second, we assess the tradeoff between accuracy and network load for the PCAg. This part involves the simulation of different network topologies in order to estimate the network loads caused by the PCAg when the routing tree changes. The results are compared with the default data collection scheme.
- Third, we assess the ability of the power iteration method (PIM) to compute the eigenvectors of a covariance matrix. This part focuses on accuracy criteria, and mainly aims at determining experimentally the convergence speed of the method. The full covariance matrix is used.
- Fourth, we assess the ability of the PCA to represent the sensor measurements when the local covariance hypothesis is applied. This part again only focuses on data accuracy, and aims at studying the loss of accuracy entailed by the local covariance hypothesis.

- Finally, we provide an experimental evaluation of the network loads caused by the computation of the eigenvectors, for both the centralized and the distributed approach.

6.1 Network simulation

Our aim in this chapter is to provide a preliminary analysis of the ability of the DPCA to capture information from sensor network data. We rely for these experiments on a simple network model, which allows us to focus on data related performance criteria. In terms of sensor networking, we assume that the sensor nodes can communicate if they are within a distance r of one another, and that they are connected to the base station by means of a routing tree. The root node is defined as the closest node to the base station. The tree is then built by iterating the two following steps. First, the set of all nodes within a distance r of the nodes already connected is selected. Second, these nodes are connected to the routing tree in such a way that the number of hops between them and the base station is minimized. The procedure stops when all nodes are connected.

Figure 6.1 gives an illustration of a network of 9 nodes arranged in a grid, where the communication range is such that all nodes connected by dashed lines can communicate. The sensor node 9 takes the role of the root node. Then, sensor nodes 8 and 6 connect to sensor 9. The terms *parent* and *children* will be used to refer to the relative places of the sensor nodes in the tree. In this example, sensor 9 is the parent of sensors 8 and 6, and sensors 8 and 6 are the children of sensor 9. If multiple parents are possible for a node, one of them is chosen arbitrarily. This is the case at step three for example, where sensor 5 can choose sensor 6 or 8 as its parent. Finally, at the fifth step, all nodes are connected. For the sake of the analysis, we will assume that transmissions and receptions are error-free, and that the tree topology remains the same over time.



(a) Step 1: node 9 is the root node.

(b) Step 2: Nodes 8 and 6 join.

(c) Step 3: Nodes 3,5 and 7 join.

(d) Step 5: All nodes are connected.

Figure 6.1: Steps of the construction of the routing for a network of 9 sensor nodes arranged in a grid. The radio range is such that all nodes connected by dashed lines can communicate.

In our experimental results, we rely on the analyses carried out in Sections 5.3 and 5.5 to estimate the accuracy and network loads of the different approaches. In particular, the accuracy of the PCAg is assessed using the NMSE criterion (Equation (5.2)), and the highest network loads are quantified using formulas summarized in Tables 5.1 and 5.2. The analysis of the role of the neighborhood size $|\mathcal{N}_i|$ and of the number of children $|\mathcal{C}_i|$ of the node in the network loads will be studied by varying the radio range r of the sensors.

6.2 Illustrative scenario - Image data set

The data used for this first set of experiments is motivated by the following scenario. Let us assume a grid of sensors that monitors an environment in which different types of phenomena may occur. This could be for example vibration sensors embedded in a bridge, recording the vibration patterns provoked by passing vehicles, or temperature sensors on an engine, recording its heat patterns over time. In both cases, it can be assumed that similar measurement patterns are expected to occur over time. In the former case, these would correspond to the set of vehicles that may cross the bridge, e.g. cars, trucks or coaches. In the latter case, the engine is likely to work in a finite set of regimes.

By relying on the analogy between a camera and a grid of sensors monitoring a phenomenon in an environment, the set of measurements collected by the sensor field at a given time instant can be seen as an *image*. We therefore create the following data set, where the spatiotemporal phenomenon monitored consists in the cyclic appearance and disappearance of three images. They visually corrrespond to the letters “W”, “S” and “N”, see Figure 6.2. The size of the images is of $100 * 100$ pixels.

An appearance/disapperance cycle is 20 epochs long, starting from a monochrome image, increasing in contrast until the 10th epoch, and then decreasing in contrast until total disappearance. The letters as seen in Figure 6.2 are maximum contrast images, presented at the 10-th epoch of a cycle. 30 cycles were generated, in which the three letters appear cyclically in the same order, thus yielding a data set of 600 observations for 100 sensor nodes.

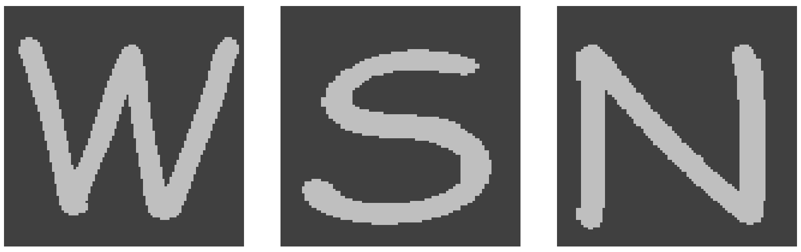
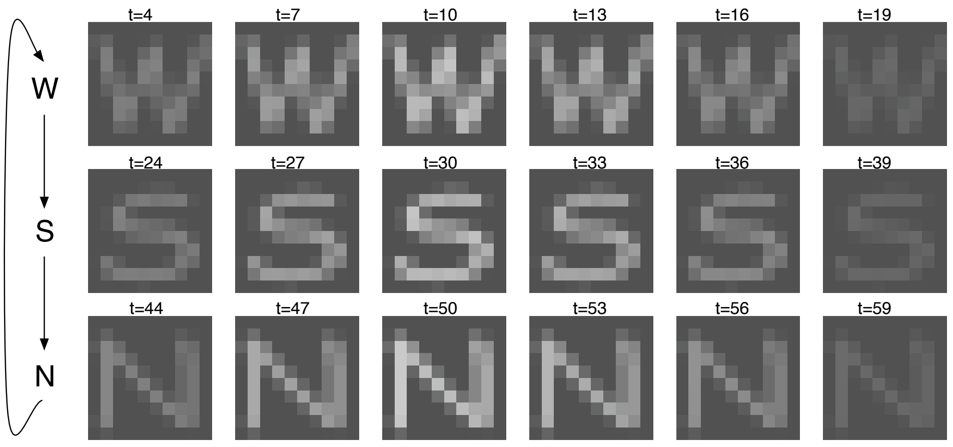


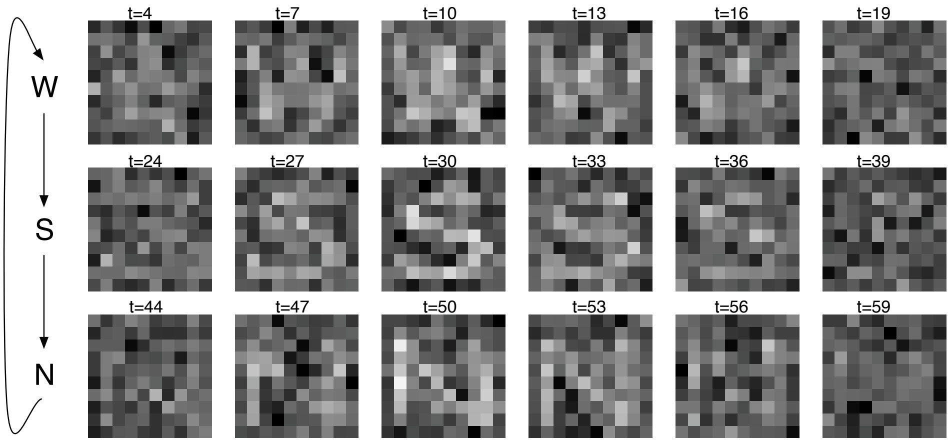
Figure 6.2: Three different phenomena, that appear and disappear in the sensed environment.

Four different levels of Gaussian noise with variance σ_n are added to this sequence of 600 observations. Denoting by σ_s the variance of the sensor measurements in the data set, the noise variance is computed with $\sigma_n = \frac{\sigma_s}{SNR}$, where SNR stands for signal-to-noise ratio. The SNR values are set to infinity (no noise), 5, 1 and 0.5, which finally yields a set of four data sets which are referred to as SNRinf, SNR5, SNR1 and SNR05, respectively.

The simulation involves a grid of $10 * 10$ sensors, uniformely distributed over the environment, capturing the average intensity of the pattern at their location (i.e. the average of the $10 * 10$ pixels square surrounding each sensor). An illustration of the corresponding discretization ouput is given in Figure 6.3, where the appearance and disappearance of each pattern is illlutrated for the data sets SNRInf and SNR1, for epochs 4, 7, 10, 13, 16 and 19. The distance between adjacent sensors is arbitrarily set to 10 meters.



(a)



(b)

Figure 6.3: The patterns "W", "S" and "N" appear in turn, a during a cycle. An appearance/disapperance cycle is 20 epochs long, starting from a monochrome image, increasing in contrast until the 10th epoch, and then decreasing in contrast until total disappearance. (a) No noise (SNRInf). (b) Signal to noise ratio is one (SNR1).

Principal component aggregation

The first results we present aim at illustrating the ability of the principal component aggregation to retain the signal of interest. More precisely, the questions addressed are

- How much information can the principal components retain?
- How can this amount of information be assessed?

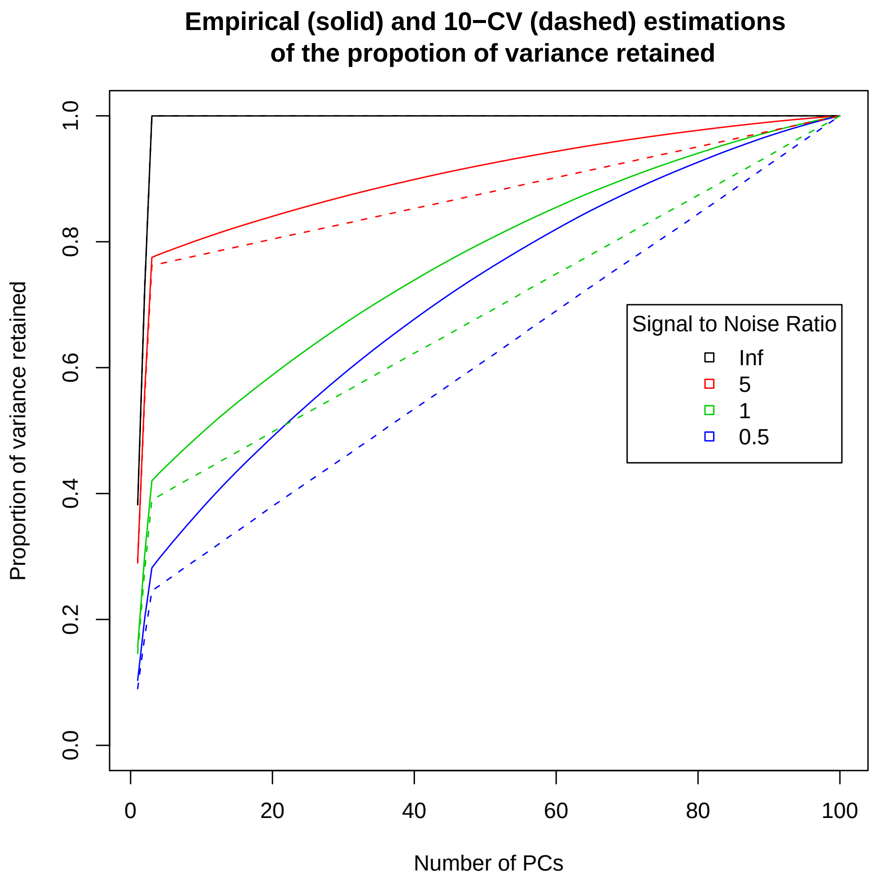


Figure 6.4: Estimation of the proportion of retained variance for different numbers of principal components and different SNRs. Solid and dashed lines represent the empirical and 10-CV estimates, respectively.

The amount of information that can be retained depends on the number of components used, and on the signal to noise ratio. The higher the number of components, the higher the amount of information retained. Similarly, the higher the signal to noise ratio, the higher the amount of information retained. This is illustrated in Figure 6.4. All the curves monotonically increase, as increasing the number of principal components necessarily increases the amount of information retained. We also observe that when the SNR increases, the amount of information retained increases as well.

The amount of information retained is assessed in terms of proportion of retained variance (cf. Section 5.3). In this data set, each vector of observations is a linear combination of one of the three patterns “W”, “S” or “N” and some white noise. Therefore, three variables completely capture the pattern of interest, and the remaining information is noise. This explains the change in the shapes of the curves from the fourth principal components. The linear trend observed from this point indicates that there is no preferential subspace for representing the variations, as they are white noise.

The solid and dashed lines differentiate the estimated proportions of retained variance, using all the data (solid, the whole 600 images are used to compute the PCs) and using

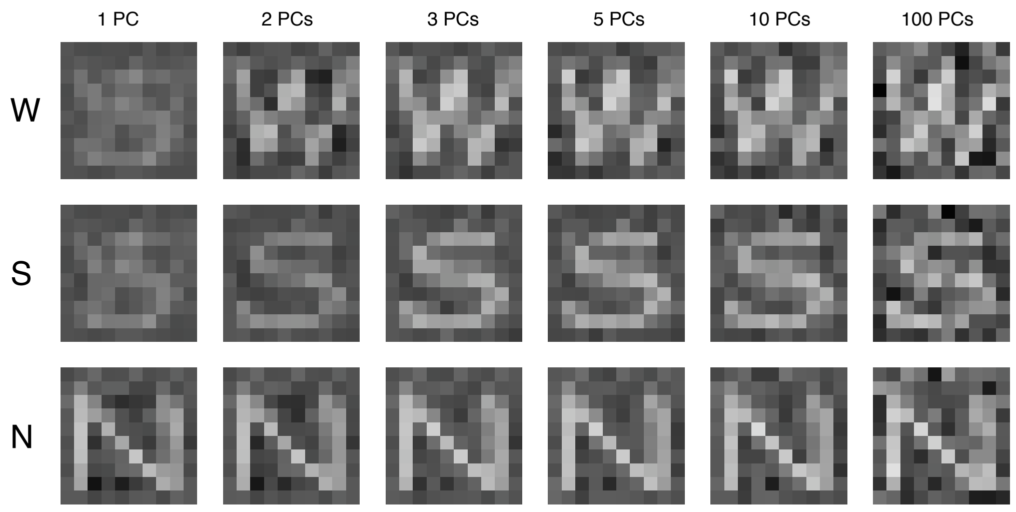


Figure 6.5: Approximation with 1, 2, 3, 5, 10 and 100 principal components for a pattern as sensed at the 10-th epoch of a cycle. The SNR is 1. 100 PCs gives the original set of measurements. Note that the PCA allows to filter noise (optimally for 3 PCs).

10-CV (dashed, only 60 images used, estimation averaged over 10 sets of 60 images). The 10-CV estimates simulate the fact that in the PCAg, the set of data used for finding the principal components is different from the data on which the PCAg is then applied. As a result, they give more realistic estimations of the expected proportions of retained variance in the application of the PCAg in real settings. We note that the differences between the empirical and the 10-CV estimations increase as the SNR decreases. This results from the poorer estimation of the eigenvectors as the amount of noise in the data increases.

Examples of the reconstruction obtained for the three patterns “W”, “S” and “N” for one, two, three, five, ten and all PCs are given in Figure 6.5. Relying on the three first PCs provides the best reconstruction of the patterns of interest. Using less components gives an approximated, compressed reconstruction of the original patterns, whereas using more components is undesirable as they recover the noise. This first example illustrates the use of the PCA for (i) compression: the PCA successfully represents the signal of interest with three variables instead of one hundred, and (ii) noise filtering: as a by-product of compression, part of the noise is removed from the raw data collected by sensors.

As mentioned in Section 2.2.4, the PCA is also useful for transforming inputs in prediction tasks in order to improve the accuracy of a learning task. We illustrate this by means of a pattern recognition task which consists in determining the type of pattern recorded by the sensor field at a given time instant.

Using the terminology of Section 2.2.1, the classification task is formulated as follows. The inputs are vectors of size k, which correspond to the projections of the set of measurements on the first k PCs. The output domain is the set {W,S,N}. The method used for learning the relation between inputs and outputs is the lazy learning [1, 6], which is a local learning technique that automatically selects the number of neighbors to use (cf. Section 2.2.3). The classification accuracies obtained for different numbers of PCs for the data set SNR5, SNR1 and SNR05 are reported in Figure 6.6. The accuracies are given in terms of percentage of correct classifications, and are assessed with a 10-fold cross validation.

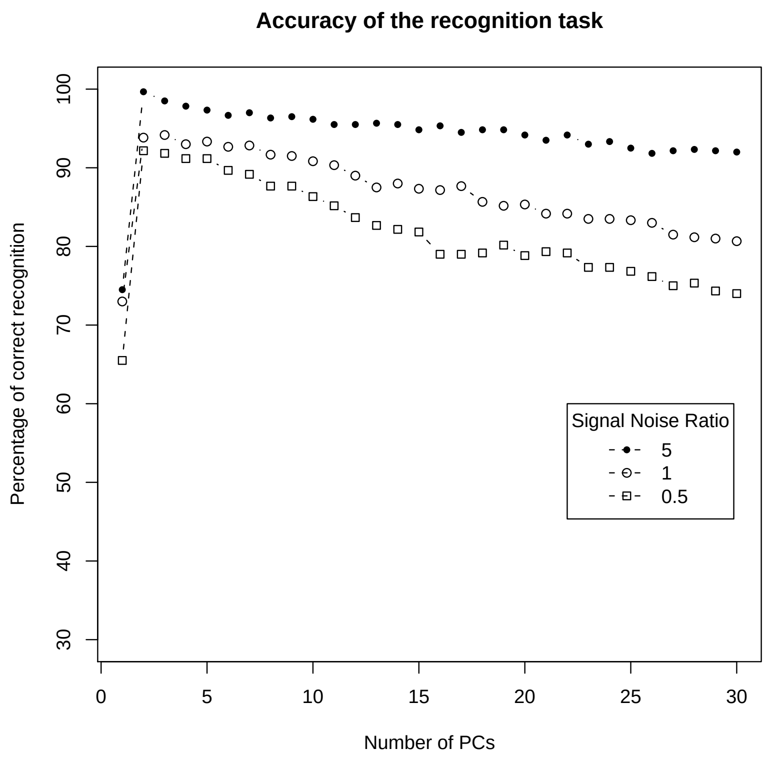


Figure 6.6: Classification accuracies in a pattern recognition task aimed at recognizing, using the PC scores, which pattern “W”, “S” or “N” is sensed by the network. 2 PC scores suffice to get almost perfect classification. The data set is SNR1.

Two PCs provide similar classification accuracies as three PCs (t-test, $p < 0.05$). This is an interesting result, as it shows that even if the subspace that contains the signal is of size 3, the classification accuracy is actually as good with only two components. The difference in accuracy between the use of one and two principal components is however important (less than 75% with one PC for about 95% of correct classifications with two PCs with SNR1 for example).

	True class W	S	N
Prediction W	112	11	30
S	82	189	33
N	6	0	137

(a) Confusion matrix for SNR1, with 1 PC.

	True class W	S	N
Prediction W	185	6	0
S	15	192	14
N	0	2	186

(b) Confusion matrix for SNR1, with 2 PCs.

Figure 6.7: Confusion matrices of the classification for SNR1 using 1 and 2 PCs.

The confusion matrices of the classifications using 1 and 2 PCs for the datset SNR1 are detailed in Tables 6.7(a) and 6.7(b). With only one PC, most of the classification errors are related to the confusion between the patterns “W” and “S”. It can be seen in Figure 6.5 that these two patterns are indeed very similar when projected on the first PC. However, given the difficulty of the task, these results are impressive. Figure 6.5 gives the reconstructions obtained at the 10-th epoch, when the contrast is the highest. Figure 6.3(b) gives an idea of the amount of noise present when the SNR is 1. For example at the fourth epoch, it is very difficult to distinguish the different patterns, and the PCA allows it. Finally, it is interesting to note that using more than 3 PCs actually decreases the classification accuracy (Figure 6.6). Therefore, collecting more data is not necessarily a way to improve a classification task. This is a result of the bias/variance tradeoff presented in Section 2.2.1.

Communication costs of the PCAg

In the following, we investigate how the PCAg can reduce the communication costs. Following the analysis of Section 5.3, the main metric used is the highest network load, defined in Table 5.1 for the D (default) and A (aggregation) actions. The PCAg reduces the highest network load if (cf. Equation (5.9))

$$q(|\mathcal{C}_{i_{\mathcal{C}*}}| + 1) < 2S - 1$$

where q is the number of PCs used, $|\mathcal{C}_{i_{\mathcal{C}*}}|$ is the largest number of children that a node has in the routing tree, and S is the network size.

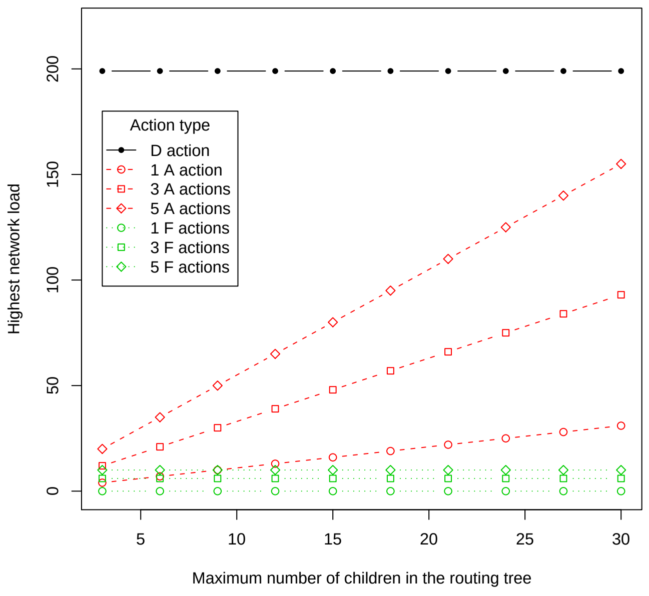


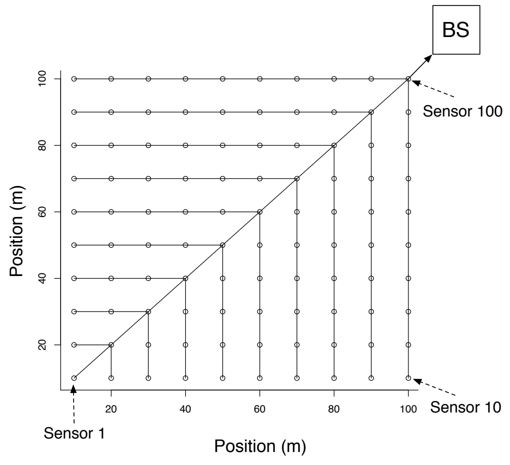
Figure 6.8: Highest network load for D, A and F actions, as a function of the largest number of children existing in the routing tree.

In the case of a D action, the highest network load is $2S - 1 = 199$, and is reported by the black solid line in Figure 6.8. For A actions, the highest network load is reported by the red dashed lines, which show the linear dependency of the load on the number of children. The circle, square and diamonds symbols are used to quantify the network loads for the aggregation of 1, 3 and 5 principal components, respectively.

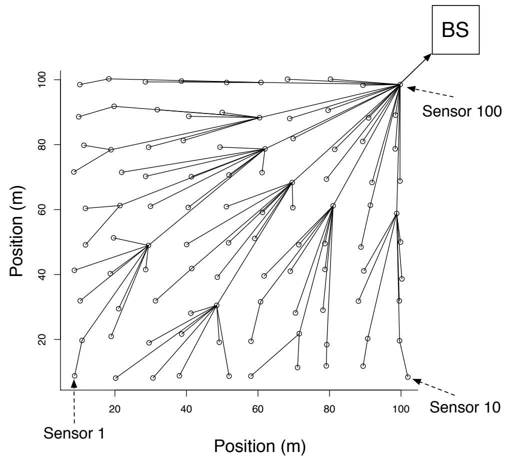
Finally, the green dotted lines quantify the highest network loads for F actions, which consist in providing all the sensors with an information issued by the base station. We included them in the same figure to give a sense of their costs compared to D and A actions. Feedback actions do not depend on aggregation, and are therefore independent of the strategy used for aggregating data. As detailed in Section 5.3, we assume that their costs was 1 for leaf nodes in the routing tree, and 2 for other nodes (as leaf node do not need to transmit data since that they do not have children). In terms of highest network load, the cost of feedback actions is low compared to A and D actions. Indeed, the maximum is 2 for one F action, or 6 for 3 F actions. Similarly to A actions, circle, square and diamonds symbols are used to quantify the load for the feedback of 1, 3 and 5 piece of data, respectively.

The main interest of this figure is to compare the communication costs of D and A actions. In the present scenario, only 3 PCs suffice to represent the signal of interest. The computation of 3 PCs requires 3 A actions, and therefore the aggregation scheme is much more efficient than the default data gathering scheme. Indeed, a D action implies to retrieve the measurements from one hundred nodes at the base station. The figure illustrates that aggregation is truly efficient.

Given the linear dependency of the network loads with respect to the number of PC scores and the number of children (cf. Equation (6.2)), it is easy to extrapolate this figure to other scenarios. For example, with an upper bound of 18 children in the routing tree, the aggregation of up to ten PC scores would still be more efficient than the default scheme $(10 * (18 + 1) < 199)$.



(a)



(b)

Figure 6.9: Routing trees obtained using a shortest path metric. Increasing the radio rage typically increases the number of children in the routing tree. (a) The radio range is of 15 meters, and the maximum number of children is 3. (b) The radio range is of 45 meters. A slight jitter is added to the sensor positions to help visualize the connections between aligned nodes. The maximum number of children is 18 (root node).

We illustrate in Figure 6.9(a) and 6.9(b), using the shortest path metric (cf. Section 6.1), examples of routing trees obtained with a radio range of 15 meter and 45 meters, respectively. With a radio range of 15 meters for example, the maximum number of children is 3. Increasing the radio range typically increases the number of children in the routing tree. For a radio range of 45 meters, the largest number of children is 18 (root node).

Distributed computation of the PCs

This section investigates the accuracy of the distributed power method for computing the principal components. Two main criteria are considered: speed of convergence and normalized mean squared error (NMSE). The speed of convergence is an important factor as each iteration of the power method is expensive in terms of communication. The NMSE is defined as the average squared error on the reconstructed samples normalized with the variance of the associated samples, cf. Equation (5.1), and is used to quantify the accuracy of the estimated eigenvectors. The computation of eigenvectors is a difficult algebra problem, and the result obtained with the PIM are compared to the QR algorithm, considered as one of the most accurate technique to compute the eigendecomposition of a matrix [91]. The NMSE of the QR and PIM methods are denoted QR NMSE and PIM NMSE, respectively.

For the PIM method, we define the speed of convergence (*Time conv*) as the number of iterations needed by the PIM to estimate an eigenvector. The convergence criteria used by the PIM are the maximum number of iterations $t_{\max}$ or a bound δ on the difference $\|v[t+1] - v[t]\|$ (cf. Algorithm 5.4). In the following experiments, the bound on the norm is set at 10^{-3}, and two maximum numbers of iterations of 50 and then 10 are assessed, respectively. All results are assessed by means of a 10-CV.

Convergence speed and accuracy are computed for the four data sets SNRInf, SNR5, SNR1 and SNR05. The results for SNRInf and SNR1 are reported in Figure 6.10 and 6.11, respectively, and the results for SNR5 and SNR05 are in Appendix D. The conclusions that can be drawn are summarized by the following observations.

$t_{\max} = 50$					
	1 PC	2 PCs	3 PCs	4 PCs	5 PCs
QR NMSE	0.618	0.271	0.000	0.000	0.000
PIM NMSE	0.618	0.271	0.000	0.000	0.000
Time conv	38.000	34.600	2.000	49.900	49.700
$t_{\max} = 10$					
	1 PC	2 PCs	3 PCs	4 PCs	5 PCs
QR NMSE	0.618	0.271	0.000	0.000	0.000
PIM NMSE	0.620	0.312	0.000	0.000	0.000
Time conv	10.000	10.000	2.000	10.000	10.000

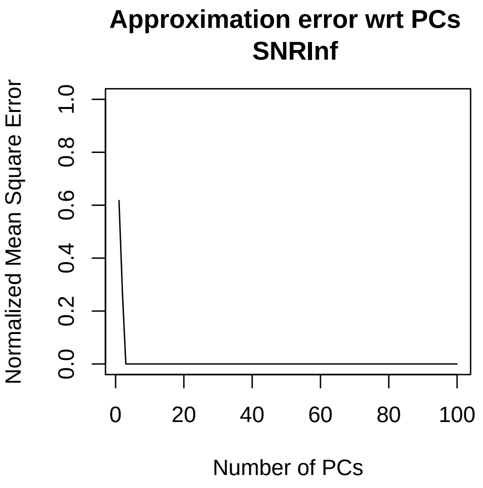


Figure 6.10: Comparison of the QR and PIM methods in terms of NMSE. The table (left) gives the NMSE for one up to five PCs, and the convergence times for the PIM method. The right figure gives a visual plot of the NMSE for the QR method. The data set is SNRInf, therefore 3 PCs allow to represent all the variations.

- A maximum number of iterations of 50 is enough for the power iteration method to converge (Estimated NMSEs are almost equal between QR and PIM). Reducing this bound to 10 implies that the convergence may not be reached. This is the case for

$t_{\max} = 50$					
	1 PC	2 PCs	3 PCs	4 PCs	5 PCs
QR NMSE	0.859	0.722	0.615	0.608	0.602
PIM NMSE	0.859	0.722	0.615	0.608	0.602
Time conv	41.300	36.700	4.800	47.900	46.700
$t_{\max} = 10$					
	1 PC	2 PCs	3 PCs	4 PCs	5 PCs
QR NMSE	0.859	0.722	0.615	0.608	0.602
PIM NMSE	0.862	0.744	0.615	0.608	0.602
Time conv	10.000	10.000	5.200	10.000	10.000

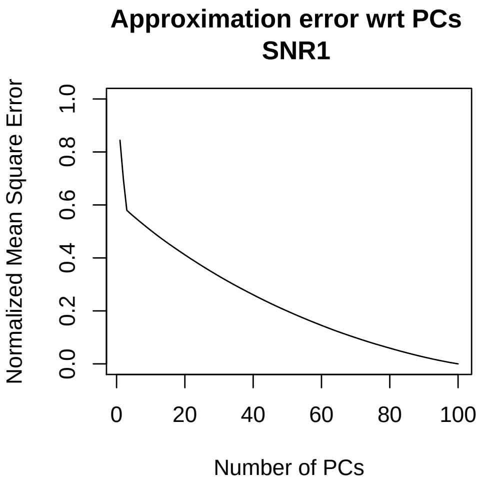


Figure 6.11: Comparison of the QR and PIM methods in terms of NMSE. The table (left) gives the NMSE for one up to five PCs, and the convergence times for the PIM method. The right figure gives a visual plot of the NMSE for the QR method. The data set is SNR1, and therefore the variations remaining from the fourth PC are noise.

the computation of the first and second PC for example. In such case, the NMSE is greater for the PIM than for the QR method. The computation of additional PCs can however allow to obtain a basis as good as the set of eigenvectors computed by the QR method. This is observed here as the computation of the third PC gives a basis whose NMSE equals the QR NMSE.

- The speed of convergence depends on the ratio of the eigenvalues. The three first PCs carry similar a amount of information, and therefore have similar eigenvalues. From the fourth PCs, eigenvalues quantify the noise, and there is therefore a high difference between the third and fourth eigenvalues. This explains why the convergence time of the third PC is much lower.

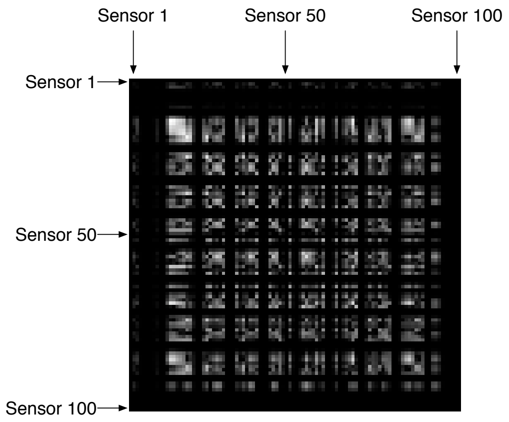
- As discussed in Section 5.4, the PIM method may be slow at converging to the best eigenvector when eigenvalues are not well separated. It however quickly converges to a subspace formed by the eigenvectors whose eigenvalues are close. This is what really matters in the PCA, as the goal is to find vectors that characterize the subspace where eigenvalues are the highest. Therefore, the PIM NMSE is close to the QR NMSE even with an early stopping of the procedure with $t_{\max} = 10$, and can "catch up" with the QR method when more PCs are computed. For all data sets (Figures 6.10 and 6.11, as well in in Appendix D), the PIM NMSE is the same as the QR NMSE for 3 PCs.

Local covariance hypothesis

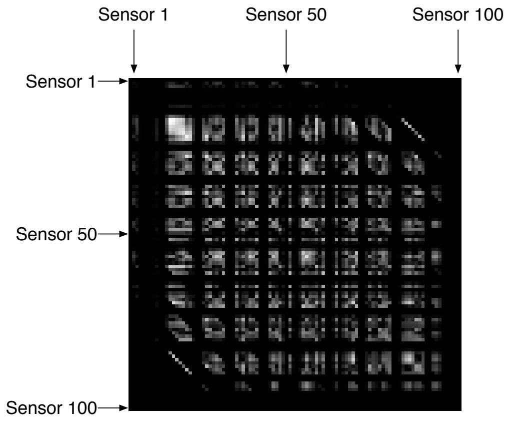
This section discusses the loss of accuracy caused by the local covariance hypothesis. We simulated the use of this hypothesis by setting the radio range r at 15, 40 and 70 meters, and by setting to zero the covariance of all sensors that were more than r meters away. With

$r = 15$, each node can only communicate with its eight immediate neighbors in the sensor grid.

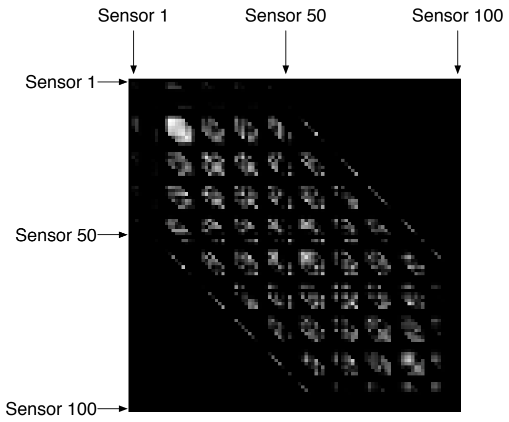
Figure 6.12(a) gives a graphical representation of the full covariance matrix for SNRInf. The sensor nodes are identified by their location in the sensor grid, cf. Figure 6.9. The covariances are normalized between 0 and 1 and represented by gray levels. The blocks appearing in this representation are explained by the two following reasons. First, sensors close geographically are not necessary next to each other in this graphical representation. Given the ordering of sensors in Figure 6.9, sensors 1 and 11 are close, but separated by ten units in the figure. Second, the monitored phenomenon is not perceived by some of the sensor nodes on the sides of the network, which makes their covariances with other sensors null.



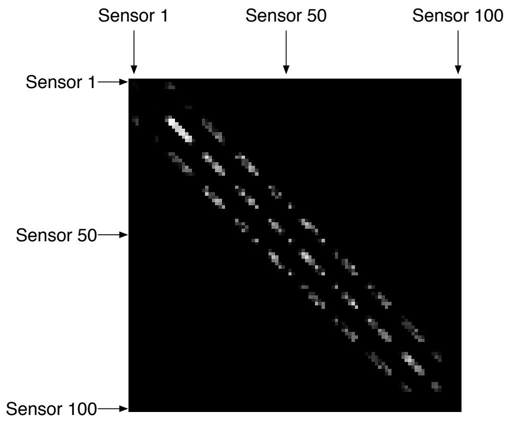
(a)



(b)



(c)



(d)

Figure 6.12: Covariances matrices resulting from the local covariance hypothesis, with the data set SNRInf. (a) Full covariance matrix. (b) Radio range 70 meters. (c) Radio range 40 meters. (d) Radio range 15 meters.

The graphical representations of the incomplete covariance matrices with $r = 70$, $r = 40$

and $r = 15$ meters are reported in Figure 6.12(b), Figure 6.12(c), Figure 6.12(d), respectively. As can be seen by comparing Figures 6.12(a) and 6.12(b), the local covariance hypothesis is not satisfied here, as distant sensor nodes (for example sensor nodes 1 to 10 and sensor nodes 90 to 100) have correlated measurements. The range $r = 15$ meters leads to a very incomplete covariance matrix. Figure D.3 in Appendix D gives similar graphical representations for the data set SNR1.

We compare in Figure 6.13 the NMSEs of the PIM and QR methods, for r in $\{15, 40, 70\}$, using the data set SNRInf. Results for SNR1 are presented in Figure 6.14, and are reported in the same form as Figure 6.10. The lower curve in the left plot serves as a reference. It gives the QR NMSEs with the full covariance matrix, and is the same as in Figure 6.10. Other curves give the QR NMSEs for different radio ranges, and the obtained NMSE with a random basis.

Radio range 40 meters					
	1 PC	2 PCs	3 PCs	4 PCs	5 PCs
QR NMSE all	0.618	0.271	0.000	0.000	0.000
QR NMSE	0.720	0.414	0.170	0.113	0.106
PIM NMSE	0.706	0.438	0.153	0.091	0.074
Radio range 15 meters					
	1 PC	2 PCs	3 PCs	4 PCs	5 PCs
QR NMSE all	0.618	0.271	0.000	0.000	0.000
QR NMSE	0.848	0.763	0.613	0.510	0.457
PIM NMSE	0.829	0.753	0.527	0.342	0.276

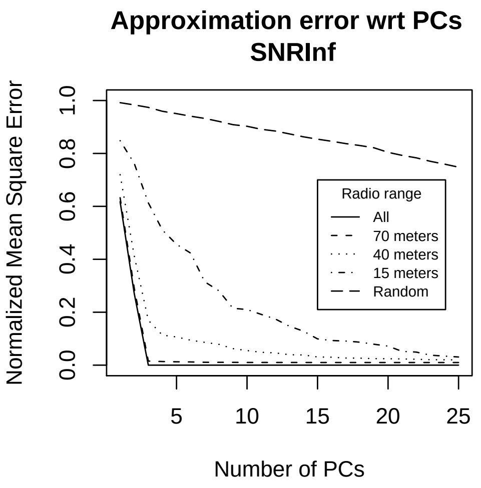


Figure 6.13: Comparison of the QR and PIM methods in terms of NMSE, when the local covariance hypothesis is applied. The table (left) gives the NMSE for one up to five PCs. Note that PIM NMSE is typically lower than NMSE as radio range decreases. The right figure gives a visual plot of the NMSE for the QR method (local covariance hypothesis applied). The data set is SNRInf.

The left table details the NMSEs for the QR and PIM approaches, for which the convergence criteria are set to $\delta = 0.001$ and $t_{\max} = 10$. We summarize the results by the following points:

- In the present scenario, the local covariance hypothesis is not valid since the patterns “W”, “S” and “N” imply that some pairs of distant sensors are correlated. As a result, the NMSE of the reconstruction increases as the radio range decreases.
- In the extreme case of $r = 15$, where sensor nodes can only communicate with their eight closest neighbors, the NMSE is noticeably higher. The subspace spanned by the estimated eigenvectors however still preserve a substantial amount of variance compared to a random basis.
- Is it worth noting that the accuracy of the PIM method is in most cases better that the QR method. This is due to the fact that with an incomplete covariance matrix, the

Radio range 40 meters					
	1 PC	2 PCs	3 PCs	4 PCs	5 PCs
QR NMSE all	0.859	0.722	0.615	0.608	0.602
QR NMSE	0.888	0.766	0.668	0.646	0.637
PIM NMSE	0.886	0.775	0.664	0.636	0.626
Radio range 15 meters					
	1 PC	2 PCs	3 PCs	4 PCs	5 PCs
QR NMSE all	0.859	0.722	0.615	0.608	0.602
QR NMSE	0.932	0.891	0.836	0.802	0.766
PIM NMSE	0.915	0.837	0.793	0.757	0.727

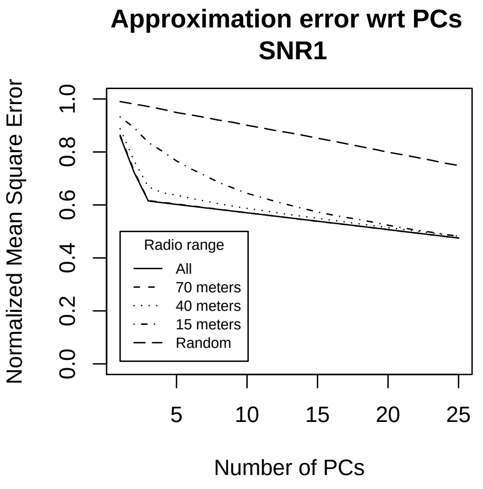


Figure 6.14: Comparison of the QR and PIM methods in terms of NMSE, when the local covariance hypothesis is applied. The table (left) gives the NMSE for one up to five PCs. Note that PIM NMSE is typically lower than NMSE as radio range decreases. The right figure gives a visual plot of the NMSE for the QR method (local covariance hypothesis applied). The data set is SNR1.

vectors that best represent the data are not the eigenvectors of the incomplete matrix, but some *nearby* vectors. Although further theoretical analysis is required to better explain these observations, the PIM method seems to perform well with incomplete covariance matrices.

The results presented so far have illustrated that the power iteration method typically converges rather fast, and that after 10 iterations a good estimate of a PC can be obtained. Also, the results obtained concerning the local covariance hypothesis are encouraging. Despite an inevitable loss due to the approximation of the covariance matrix, the PCs computed are observed to retain much more variance than a random basis.

Communication costs related to the computation of the PCs

This section investigates the highest network loads (HNL) implied by the computation of the PCs, and more particularly compare the centralized approach to the distributed one. We rely on the formulas derived in Table 5.2, namely,

- HNL of the centralized approach for computing the sample covariance matrix:

$$L_{\max}^{\text{Cov}_{\text{cent}}} = N(2S-1)$$

- HNL of the centralized approach for computing the eigenvectors:

$$L_{\max}^{\text{EV}_{\text{cent}}} = 2qS$$

- HNL of the distributed approach for computing the sample covariance matrix (with the local covariance hypothesis

$$L_{\max}^{\mathrm{Cov}_{\mathrm{dist}}} = N|\mathcal{N}_{i^*_{\mathcal{N}}}|$$

- HNL of the distributed approach for computing the eigenvectors:

$$L_{\max}^{\mathrm{EV}_{\mathrm{dist}}} = \max_i \sum_{k=1}^{q} t_{\mathrm{conv}}(k)(1 + |\mathcal{N}_i| + k(|\mathcal{C}_i| + 2 + 1(|\mathcal{C}_i| \neq 0))$$

to estimate the nHNL implied by the two approaches. A fair comparison between the two approach is difficult as they do not rely on the same parameters. For the centralized approach, the parameters influencing the HNL are

- the number S of sensors in the network, and
- the number N of epochs used to estimate the covariance matrix, and
- the number q of principal components to compute.

In the distributed approach, the HNL depends on N, q, and additionally on

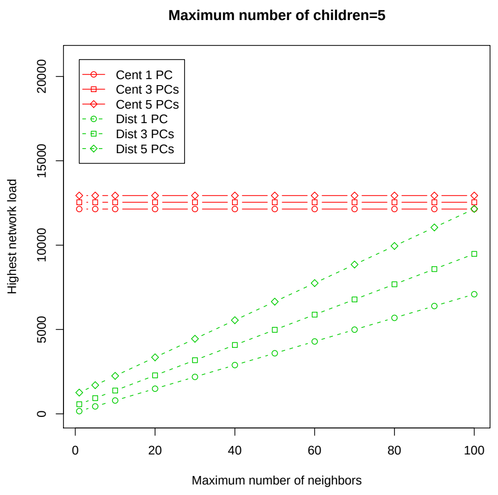
- the largest number of children $|\mathcal{C}_{i^*_{\mathcal{C}}}|$ in the routing tree,
- the largest neighborhood size $|\mathcal{N}_{i^*_{\mathcal{N}}}|$ in the network, and
- the convergence times $t_{\mathrm{conv}}(k)$ of the power iteration method for each PC computed, $1 \leq k \leq q$.

As a baseline for the comparison, we analyze the HNL using the parameter values that were used in all the previous experiments. First, the number of samples used to estimate the sample covariance matrix was set to 60 (implicitly, with the 10-fold cross validation). The number of sensors was 100, and the number of components computed was ranged from 1 to 5. Finally, the maximum number of iterations was set to 10, and we therefore set $t_{\mathrm{conv}}(k) = 10, \forall k$, which is a worst case estimate for the HNL caused by the distributed approach.

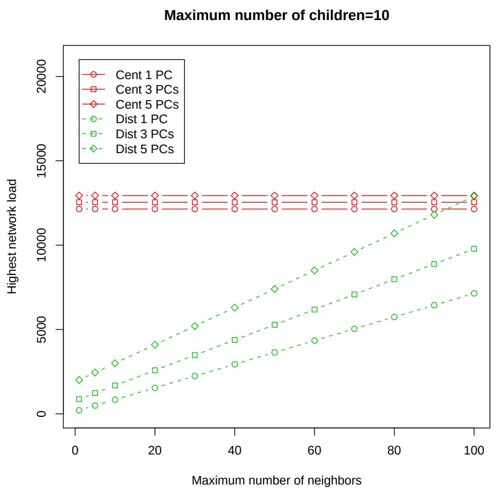
Using these parameters, we report in Figure 6.15 the highest network loads caused by the computation of 1 (circles), 3 (square) and 5 (diamonds) PCs, as a function of the largest neighborhood size $|\mathcal{N}_{i^*_{\mathcal{N}}}|$. The influence of the parameter $|\mathcal{C}_{i^*_{\mathcal{C}}}|$ is illustrated by reporting the highest network loads for $|\mathcal{C}_{i^*_{\mathcal{C}}}| = 5$ (Figure 6.15(a)) and $|\mathcal{C}_{i^*_{\mathcal{C}}}| = 10$ (Figure 6.15(b)). A breakdown of the loads into the matrix computation stage and the eigenvector computation stage is given in Table 6.1.
We summarize these results by the following points:

- In the centralized approach, most of the network load is caused by the collection of N vectors of measurements at the base station (cf. Table 6.1). The tradeoff caused by this parameter can easily be seen by in Figure 6.15. The HNL for the centralized approach is independent of $|\mathcal{N}_{i^*_{\mathcal{N}}}|$, and depends linearly on N. Therefore, increasing or decreasing$N$ (here $N = 60$) makes the horizontal lines go down or up.

- In the distributed approach, the parameter $|\mathcal{N}_{i^*_{\mathcal{N}}}|$ plays an important role, as it trades network load for accuracy. The lower its value, the lower the network load entailed



(a) Highest number of children $|\mathcal{C}_{i^*_\mathcal{C}}| = 5$.



(b) Highest number of children $|\mathcal{C}_{i^*_\mathcal{C}}| = 10$.

Figure 6.15: Highest network loads entailed by the computation of different numbers of PCs, for the centralized (solid lines) and distributed (dashed lines) approaches. The sizes of the largest neighborhood size $|\mathcal{N}_{i^*_\mathcal{N}}|$ and largest number of children $|\mathcal{C}_{i^*_\mathcal{C}}|$ play a role in the network loads caused by the distributed approach.

5 children							10 children					
	5	10	20	40	60	100	5	10	20	40	60	100
$L_{\max}^{Cov_{cent}}$	11940	11940	11940	11940	11940	11940	11940	11940	11940	11940	11940	11940
$L_{\max}^{Cov_{dist}}$	300	600	1200	2400	3600	6000	300	600	1200	2400	3600	6000
$L_{\max}^{EV_{cent}}(1)$	200	200	200	200	200	200	200	200	200	200	200	200
$L_{\max}^{EV_{cent}}(3)$	600	600	600	600	600	600	600	600	600	600	600	600
$L_{\max}^{EV_{cent}}(5)$	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
$L_{\max}^{EV_{dist}}(1)$	140	190	290	490	690	1090	190	240	340	540	740	1140
$L_{\max}^{EV_{dist}}(3)$	630	780	1080	1680	2280	3480	930	1080	1380	1980	2580	3780
$L_{\max}^{EV_{dist}}(5)$	1400	1650	2150	3150	4150	6150	2150	2400	2900	3900	4900	6900

Table 6.1: Breakdown of the loads into the matrix computation stage and the eigenvector computation stage, for the centralized and distributed approaches.

by the computation of the PCs. However, low values imply that most covariances are assumed to be null (local covariance hypothesis) during the covariance matrix computation stage. As a result, the PCs computed may lead to higher loss than the PCs obtained by the centralized approach. A low radio range of 15 meters provides strong reduction of the highest network load (Figure 6.15), but NMSE much higher than the centralized scheme (NMSE of 0.527 with 3 PCs - cf. Figure 6.13). On the opposite, a radio range allowing all sensor nodes to communicate provides NMSEs as low as the centralized approach, but at HNL that can be higher than the centralized approach (and this depends on N).

- The HNL of the distributed approach increases quadratically with the number of PCs computed, which can be seen as the slope of the green dashed lines in Figure 6.15 increases with the number of PCs. The distributed approach is therefore likely to be more efficient when the number of PCs to compute is low.

- Finally, the quadratic increase of the slope is proportional to the quantity $|\mathcal{C}_{i^*_{\mathcal{C}}}|$. It is worth mentioning that this quantity depends on the structure of the routing tree, and that a possible optimization for the DPPC is to combine it with a routing tree algorithm that minimizes the largest number of children in the tree.

These different tradeoffs show that the distributed approach can reduce the highest network load in certain cases. In particular, the approach can provide important communication savings when the number of PCs to compute is low, or when the number of observations required to compute the sample covariance matrix is large.

6.3 Data from Intel Berkeley laboratory

The data set used in this second set of experiments consists in temperature measurements, which come from a deployment of 54 sensors in the Intel research laboratory at Berkeley. The deployment took place between February 28th and April 5th, 2004. A picture of the deployment is provided in Figure 6.16, where sensor nodes are identified by numbers ranging from 1 to 54.

Due to the prototypical nature of this deployment, many sensor readings were missing. We selected from this data set a subset of five days of measurements, during which almost all data are present, except for the ones collected by sensors 5 and 15. The readings were originally sampled every thirty-one seconds. A preprocessing stage where data was discretized in thirty second intervals was applied to the data set. The preprocessing was carried out using linear interpolation. After preprocessing, the data set contained a trace of 14400 readings from 52 different sensors (all but sensors 5 and 15).

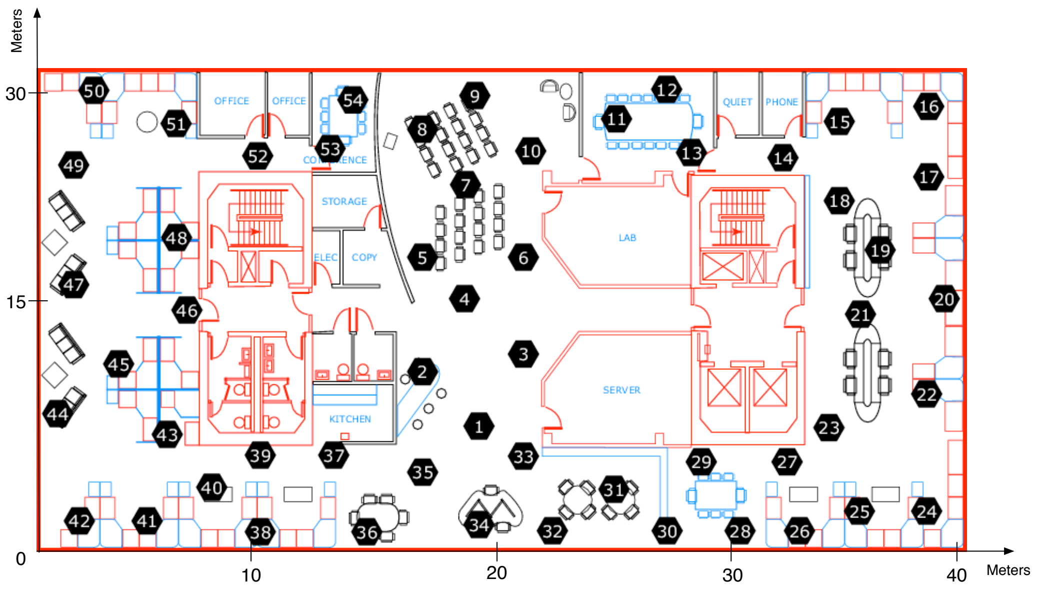


Figure 6.16: Location of the 54 sensors in the Intel Research Laboratory. The area of the laboratory is close to 1200m^2.

To give a sense of the nature of the measurements, we reported in Figure 6.17 the variations of temperature for sensors 21 and 49. Among all the pairs of sensors, the measurements of sensors 21 and 49 had the lowest correlation over the five day period considered. One can see on the deployment map Figure 6.16 they they were located on opposite sides of the laboratory.

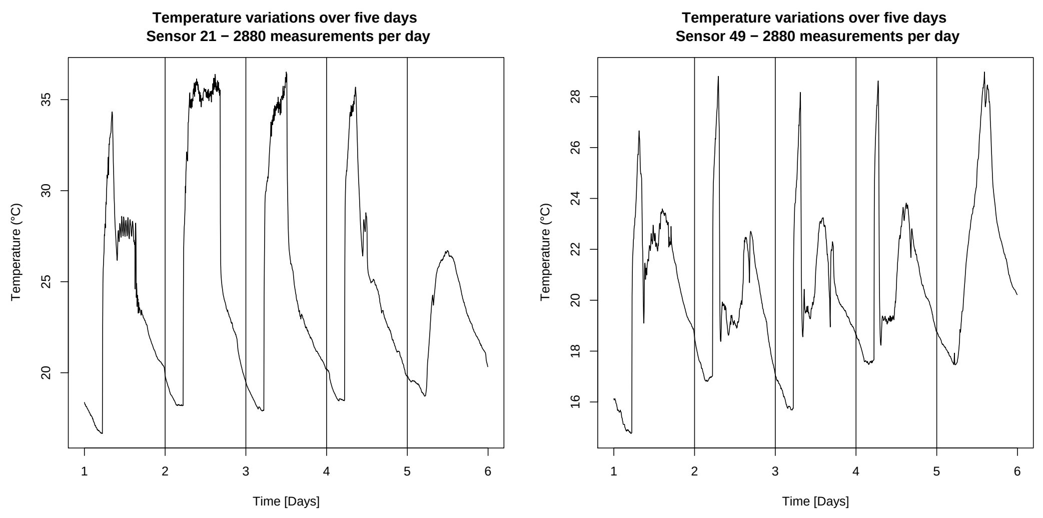


Figure 6.17: Temperature measurements collected by sensors 21 and 49 over a five day period.

As examples of the dependencies existing between sensor measurements, we reported in Figure 6.18 the dependencies existing between the measurements of sensors 21 and 49, and those between sensors 21 and 22. In terms of range, the temperature over the whole set of data ranged from about 15°C to 35°C.

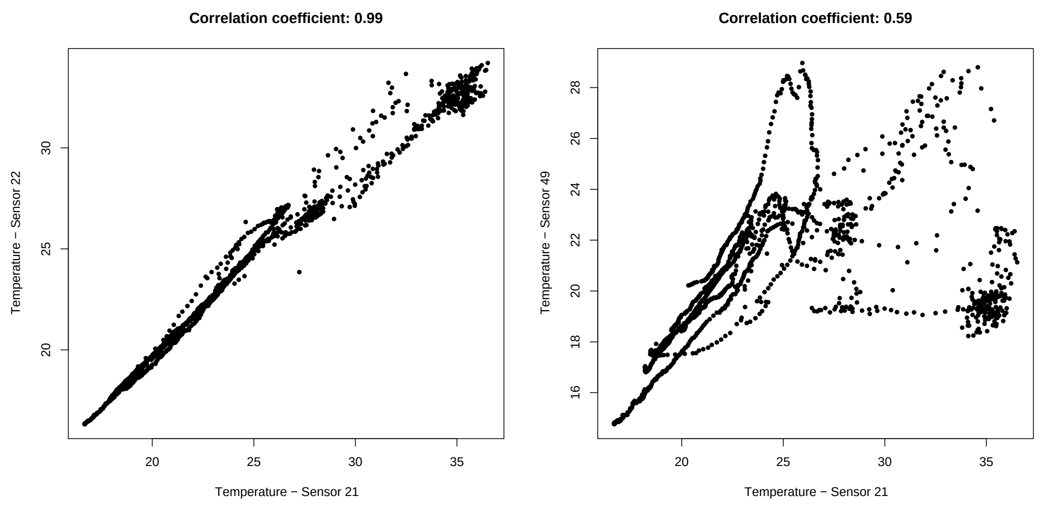


Figure 6.18: Examples of the dependencies between the measurements of sensor 21 and sensor 49.

Principal component aggregation

We first study the amount of variance that the PCA can retain in the measurements for different training periods and number of components. Figure 6.19 provides the average retained variance for the first 25 principal components. As in Section 6.2, we relied on 10-fold cross validation for assessing the percentages of retained variance. The data set is split in ten consecutive blocks of 1440 observations each, i.e., half a day of measurements. Each of the ten blocks is used in turn as a *training* set to compute the covariance matrix and its eigenvectors, while the remaining observations are used to estimate the percentage of retained variance. The ten estimates are then averaged. This simulates the fact that in practice, the use of the PCAg involves first a training stage to compute the eigenvectors, followed by an operational stage where the method is applied on new measurements.

In Figure 6.19, the upper line gives the average amount of retained variance when the principal components are computed with the test sets, and provides an upper bound on the compression efficiency that the PCA can achieve on this data set. The lower curve gives the average amount of retained variance on the test set when the components are computed with the training set. This figure shows that the first principal component accounts on average for almost 80% of the variance, while 90% and 95% of variance are retained with 4 and 10 components, respectively. The confidence level of these estimates was about $\pm 5\%$. Additional experiments were run using K-fold cross validation with K ranging from 2 to 30. The percentages of retained variance on the test data blocks tended to decrease with K. Losses of a few percents were observed for K higher than 15 (less than nine hours of data).

The amount of retained variance increases very fast with the first principal component, and becomes almost linear after about ten components. A linear increase of retained variance with the number of principal components reflects the fact that the components obtained by the PCA are actually no better than random components (cf. Section 5.3).

Figure 6.20 illustrates the obtained approximations during the first round of the cross validation (i.e., principal components are computed from the first 12 hours of measurements) for the sensors 21 and 49, using one, five and ten principal components. The first principal component allows to represent the daily increases and decreases of temperature. Increasing the number of principal components allows to better approximate the local variations. This is illustrated by sensor 49, where the temperature seems artificially maintained at 20°C around noon during the second, third and fourth day (probably due to the activation of an air conditioning system at a location close to sensor 49). Five components allow to reasonably well model this phenomenon.

The data used for training are important. This is illustrated by sensor 21, where the temperature measurements during the first day have a different evolution than the other days (sharp decrease from $34°C$ to $27°C$ at bout 10h). As a result, more components are required to properly represent the measurements during that part of the day.

Communication costs of the PCAg

We now compare the communication costs entailed by D, A, and F actions (cf. Table 5.1, Section 5.3), as a function of the largest number of children and the number of aggregated PC scores. The methodology is the same as in Section 6.2, and we rely on the highest network load to compare the costs of the different actions. The results are reported in Figure 6.21. The differences with Figure 6.8 are that

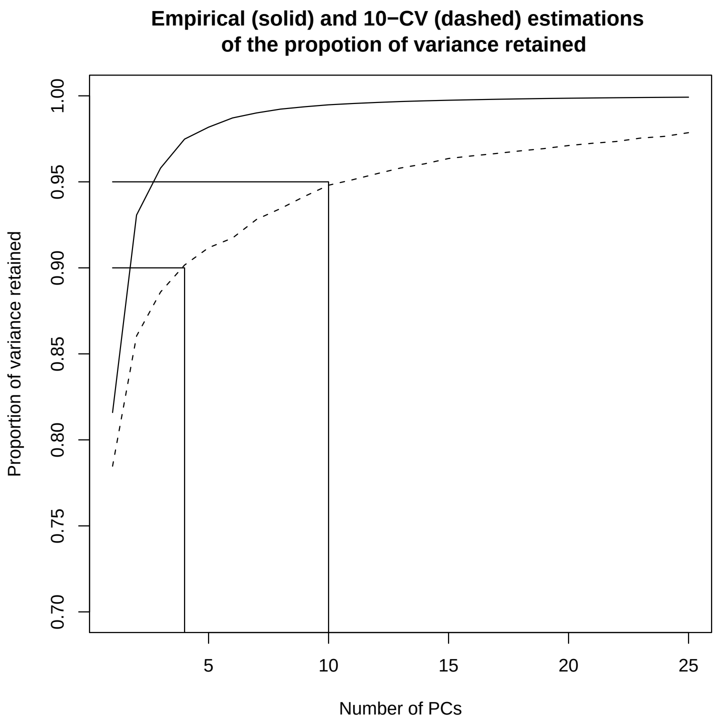


Figure 6.19: Capacity of principal components to retain the measurement variance.

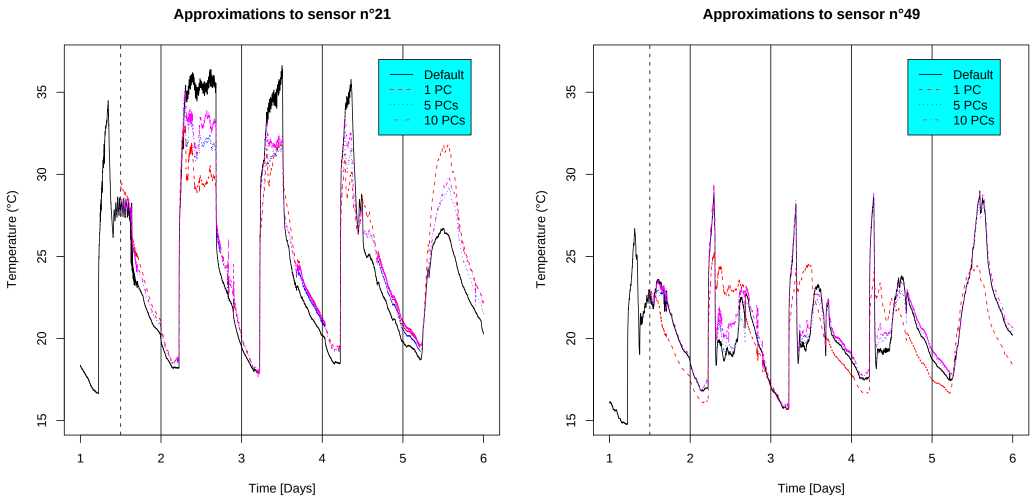


Figure 6.20: Approximations on the test set for the sensor 21 (left) and 49 (right), using one, five and ten principal components. The covariance matrix was computed from the first twelve hours of measurements (before the vertical dashed line).

- the highest network load sustained by the root node for the D action is now of $2S-1 = 103$ packets per epoch,
- the number of PC scores aggregated is of 1, 5 and 10. Note that 5 and 10 PCs retain

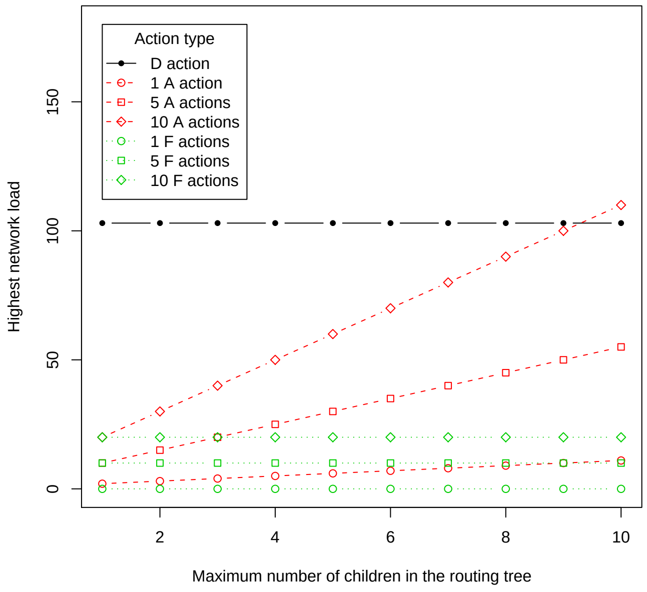


Figure 6.21: Highest network load for D, A and F actions, as a function of the largest number of children existing in the routing tree.

90% and 95% of the variance, respectively (cf. Figure 6.19).

The comments on these results are similar to those concerning the Image data set in Section 6.2. They illustrate that as long as the routing tree does not have nodes with exceedingly high number of children, the PCAg allows to reduce the highest network load.

Distributed computation of the PCs

The accuracy of the distributed power method is reported in Figure 6.22. The results are compared with the QR method, assumed to provide accurate eigenvectors. The QR NMSE is reported as a function of the number of PCs (up to 15) on the left plot. The NMSE obtained by the QR method (QR NMSE), NMSE of the power iteration method (PIM NMSE), and convergence times (*Time conv*), are reported in the table on the left. The bound δ on the difference $\|v[t+1] - v[t]\|$ is set to 0.001, and the maximum number of iteration is set to 10, as motivated by the experimental results obtained with the Image data set.

As for the results obtained with the Image data set, we observe that the PIM method allows to converge to eigenvectors that provide NMSEs very close to QR NMSEs. It is interesting to note that for the 8-th PC, the NMSE is even lower for the PIM than for the centralized QR method. This stems for the use of a 10-CV, where performances are assessed on data different from those used for the training. As a result, the 8-th approximated eigenvector turns out to be better at retaining variance in the test set than the exact one. The converse effect can also happen, as illustrated by the lower NMSE obtained by the power method after the computation of the 9-th eigenvector.

The other difference with the results obtained with the Image data set concerns the convergence times of the power method. The convergence times are in this data set much

	1 PC	2 PCs	3 PCs	4 PCs	5 PCs
QR NMSE	0.215	0.140	0.114	0.098	0.088
PIM NMSE	0.215	0.140	0.114	0.099	0.088
Time conv	2.700	4.900	7.400	8.700	9.800
	6 PCs	8 PCs	10 PCs	12 PCs	15 PCs
QR NMSE	0.083	0.066	0.052	0.045	0.036
PIM NMSE	0.082	0.063	0.053	0.045	0.036
Time conv	9.800	10.000	9.900	10.000	10.000

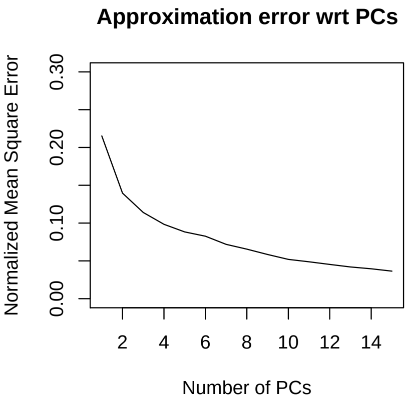


Figure 6.22: Comparison of the QR and PIM methods in terms of NMSE. The table (left) gives the NMSE for one up to fifteen PCs, and the convergence times for the PIM method. The right figure gives a visual plot of the NMSE for the QR method (optimal).

faster, especially for the first PCs (on average 2.7 and 4.9 iterations for the first and second eigenvector, respectively). This is due to the fact that the eigenvalues are more separated in this data set than in the Image data set (cf. Figures 6.19 and 6.4).

As a conclusion, these results illustrate again the ability of the power method to converge quickly and to estimate eigenvectors that efficiently span the signal subspace.

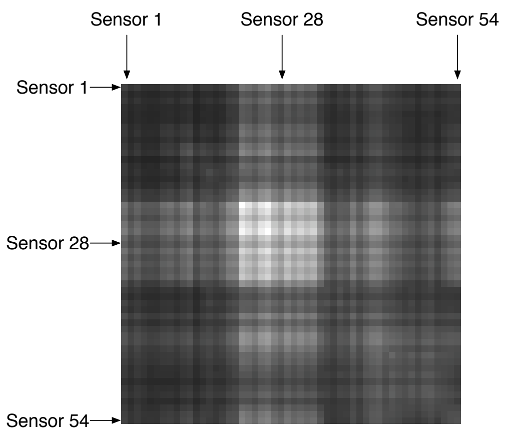
Local covariance hypothesis

This section investigates the accuracy of the eigendecomposition when the local covariance hypothesis is used. The local covariance hypothesis is tested, as in Section 6.2, by changing the radio range r. Since the network size is smaller than in Section 6.2, the tested values for the radio range are $r = 6$, $r = 10$ and $r = 20$ meters. Figure 6.23 gives graphical representations of the covariance matrices resulting from the local covariance hypothesis.

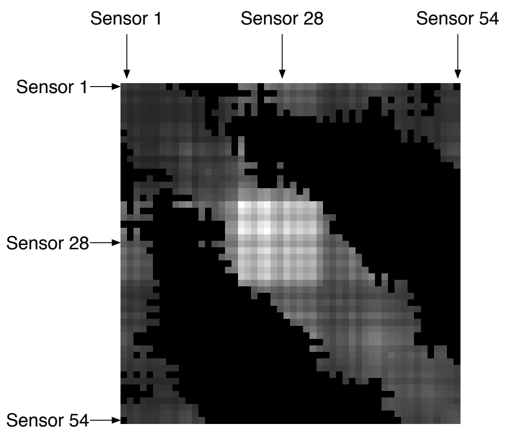
The group of sensors around sensor node 28 have higher variances and covariances than the others. This means that their range of measurements are higher, which can be for example explained by the presence of a window allowing sunlight to increase locally the temperature measurements. Figure 6.23(a) shows that the local covariance hypothesis does not hold, as there exists covariances between distant sensor measurements. A radio range of $r = 6$ meters makes the sample covariance matrix very incomplete.

Figure 6.24 presents the NMSEs of the QR and PIM methods using the full and incomplete sample covariance matrices. The results are qualitatively very similar to those obtained for the Image data set (Figures 6.13 and 6.14). For few PCs, the loss caused by the local covariance hypothesis may be sensible. For example, the NMSE is of 0.2 with one PC, full covariance matrix, and increases to 0.7 if the covariance matrix entries of sensors more than six meters apart are set to zero. As the number of PCs computed increases, the losses in NMSEs however become comparable.

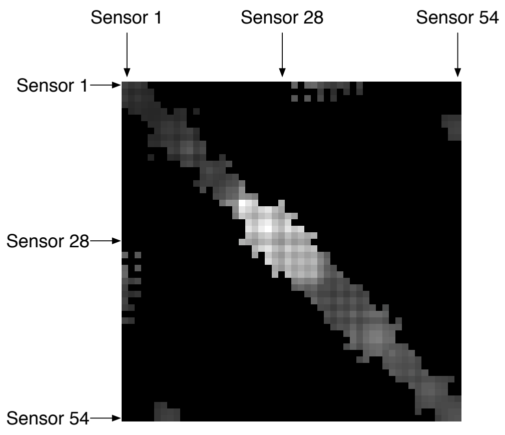
In terms of accuracy of the power method, we observe the same effect as in Section 6.2, where the NMSEs entailed by the power method may be lower than those of the QR method. For example, for the second PC, the NMSE of the power method is of 0.384, and 0.51 for



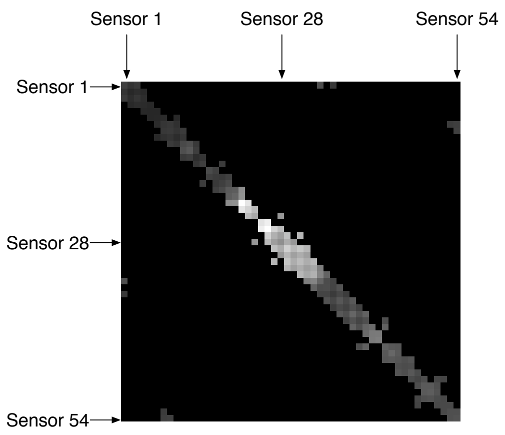
(a)



(b)



(c)



(d)

Figure 6.23: Covariances matrices resulting from the local covariance hypothesis. The covariances are normalized between 0 and 1 and represented by gray levels. The highest value is the variance of sensor 21 ($\sigma_2 = 30.8$ before normalization). (a) Full covariance matrix. (b) Radio range 20 meters. (c) Radio range 10 meters. (d) Radio range 6 meters.

the QR method (second and third line in the right Table). This is again due to the fact that the convergence is not attained for the computation of the second PC, as the upper bound of $t_{\max} = 10$ iterations is reached. As a consequence, the basis found by the PIM method differs from that obtained by the QR method. We finally observe that as the number of PCs computed increases, the NMSE of the QR method gets lower than that of the PIM method. For 10 PCs for example, the QR NMSE is of 0.079 against 0.101 for the PIM method (while the NMSE would be of 0.052 if the PC were computed from the full covariance matrix).

Radio range 10 meters

	1 PC	2 PCs	3 PCs	4 PCs	5 PCs
QR NMSE all	0.215	0.140	0.114	0.098	0.088
QR NMSE	0.547	0.510	0.380	0.299	0.213
PIM NMSE	0.547	0.384	0.289	0.193	0.146
Time conv	7.800	10.000	10.000	10.000	10.000
	6 PCs	8 PCs	10 PCs	12 PCs	15 PCs
QR NMSE all	0.083	0.066	0.052	0.045	0.036
QR NMSE	0.192	0.111	0.079	0.067	0.055
PIM NMSE	0.127	0.116	0.101	0.088	0.077
Time conv	10.000	10.000	10.000	10.000	10.000

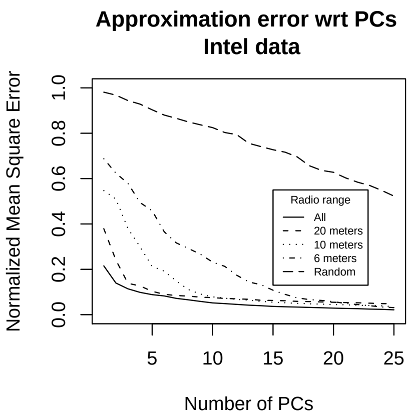


Figure 6.24: Comparison of the QR and PIM methods in terms of NMSE, when the local covariance hypothesis is applied. The table (left) gives the NMSE for one up to five PCs. Note that PIM NMSE is typically lower than NMSE when few PCs are computed. The right figure gives a visual plot of the NMSE for the QR method (local covariance hypothesis applied).

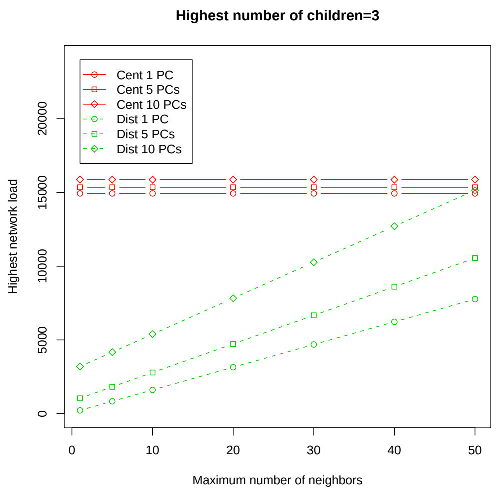
Communication costs related to the computation of the PCs

We finally compare, in a similar fashion as in Section 6.2, the communication costs entailed by the centralized and distributed approaches for computing the principal components. Given the differences between the Image and Intel data set, we adapt the following parameters:

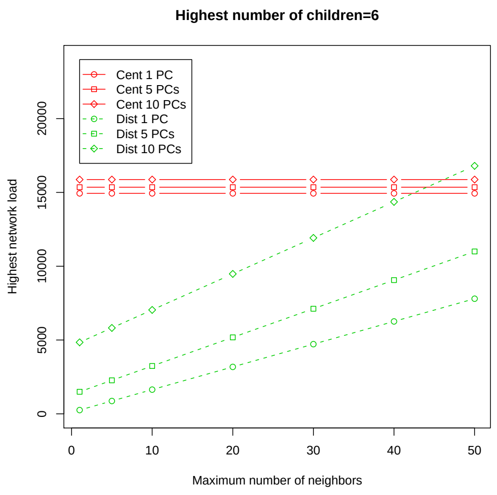
- the number of samples used for estimating the covariance is set to $N = 144$, which corresponds to half a day of measurements sampled every 5 minutes.
- the highest network loads are detailed for the computation of 1, 5 and 10 principal components, which are observed to retain around 80%, 90%, 95% of the variance, respectively.
- the neighborhood size $|\mathcal{N}_{i^*_{\mathcal{N}}}|$ takes values in the interval $[1, 50]$.
- given the smaller size of the network, we provide the details of the highest network load for $|\mathcal{C}_{i^*_{\mathcal{C}}}| = 3$ and $|\mathcal{C}_{i^*_{\mathcal{C}}}| = 6$.

As for the Image data set in Section 6.2, we report in Figure 6.25 the highest network load caused by the computation of 1 (circles), 3 (square) and 5 (diamonds) PCs as a function of the $|\mathcal{N}_{i^*_{\mathcal{N}}}|$. Solid lines represent the loads for the centralized approach, and dotted lines the loads for the distributed approach. A breakdown of these costs into the matrix computation stage, and the eigenvector computation stage is given in Table 6.2.

We make the following observations. In quantitative terms, the communication costs are more important than for the Image data set. This is due to the fact that more samples are required for computing the covariance matrix, and that more eigenvectors are computed. In qualitative terms however, the trends are very similar to those obtained for the Image data set, and the conclusion we can drawn are the same. In particular, the distributed approach



(a) 3 neighbors.



(b) 6 neighbors.

Figure 6.25: Highest network loads entailed by the computation of different number of PCs, for the centralized (solid lines) and distributed (dashed lines) approaches. The sizes of the largest neighborhood size $|\mathcal{N}_{i^*_{\mathcal{N}}}|$ and largest number of children $|\mathcal{C}_{i^*_{\mathcal{C}}}|$ play a role in the network loads caused by the distributed approach.

3 children							6 children					
	5	10	20	30	40	50	5	10	20	30	40	50
$L_{\max}^{Cov_{cent}}$	14832	14832	14832	14832	14832	14832	14832	14832	14832	14832	14832	14832
$L_{\max}^{Cov_{dist}}$	720	1440	2880	4320	5760	7200	720	1440	2880	4320	5760	7200
$L_{\max}^{EV_{cent}}(1)$	104	104	104	104	104	104	104	104	104	104	104	104
$L_{\max}^{EV_{cent}}(5)$	520	520	520	520	520	520	520	520	520	520	520	520
$L_{\max}^{EV_{cent}}(10)$	1040	1040	1040	1040	1040	1040	1040	1040	1040	1040	1040	1040
$L_{\max}^{EV_{dist}}(1)$	120	170	270	370	470	570	150	200	300	400	500	600
$L_{\max}^{EV_{dist}}(5)$	1100	1350	1850	2350	2850	3350	1550	1800	2300	2800	3300	3800
$L_{\max}^{EV_{dist}}(10)$	3450	3950	4950	5950	6950	7950	5100	5600	6600	7600	8600	9600

Table 6.2: Breakdown of the loads into the matrix computation stage and the eigenvector computation stage, for the centralized and distributed approaches.

outperforms the centralized approach if the number of observations required to compute the sample covariance matrix is large, or if the number of PCs required is low.

6.4 Summary

The experimental results presented in this chapter illustrated the ability of the PCAg to effectively represent the sensed phenomenon while dramatically decreasing the network load. The efficiency of the distributed computation of the eigenvectors in comparison to the centralized approach is however subject to caution. In particular, the quadratic cost of the distributed approach in terms of number of computed PCs makes the method rapidly inefficient if the number of computed PCs is not low enough. However, the distributed approach has the advantage of decreasing in all cases the communication costs related to the estimation of the covariance matrix. As a result, the distributed approach appears suitable if the number of observations required for the estimation of the covariance matrix is high, and if the number of PCs to compute is low.

Chapter 7

Conclusions

Environmental monitoring is a particularly exciting and promising application for sensor networks. Tiny inexpensive wireless sensors can now be densely deployed throughout an environment, enabling to observe the physical world at scales that were previously not possible. Sensor networks bring a new instrument for observing our world, which may well be in the future called a *macroscope* [86], providing a way to *see* the evolution of spatiotemporal physical phenomena over numerous types of environments.

Observing the world with a wireless sensor network however raises difficult issues, since application requirements such as accuracy, lifetime, or latency, very often conflict with the limited computational, radio throughput and energy resources of the network. This thesis investigated how techniques based on machine learning could be used to address some of these issues, and focused in particular on the tradeoff between data accuracy, energy-efficiency and network load in data collection tasks. This chapter summarizes the main results of the thesis, and opens the topic to further research tracks.

7.1 Summary of the main results

The data collected by a sensor network represents the spatiotemporal evolution of phenomena taking place in the monitored environment. Most of the time, this evolution is governed by some physical, chemical or biological laws, and is therefore to a certain extent predictable. When a prediction model can represent with sufficient accuracy the evolution of the phenomenon, then significant savings of the network resources, particularly in terms of communication and energy, can be achieved.

Adaptive Model Selection

Our first contribution in this thesis was to show that the use of prediction models can be made more efficient by combining prediction techniques with model selection techniques. The main reason stems from the fact that in most cases, no or little information is available on the characteristics of sensor measurement. We moreover showed that the efficiency of a prediction model also depends on the additional costs incurred by transmitting the model parameters to a base station.

Our algorithm, the *adaptive model selection* (AMS), deals with all these issues by considering several *candidates* for predicting sensor measurements. Each candidate is a prediction model, whose accuracy at predicting the measurements is at first unknown. The AMS then locally runs a *competition*, in which the performances of the models are estimated by the sensor node. We introduced the use of a statistical procedure called *racing*, which allows to determine, with a high confidence, the model that performs best over time. Our method was validated by simulating the algorithm on 14 time series representing various types of phenomena. In *all* 14 series, the AMS was able to select the model that provided the best prediction accuracy. We finally showed that the implementation of the AMS on real-world wireless sensor platforms could easily be done using TinyOS, the reference programming framework for wireless sensors.

Distributed Principal Component Analysis

The second contribution in this thesis was to show that the Principal Component Analysis, a versatile data modeling technique, can be efficiently implemented in a distributed manner in a sensor network. The main mathematical trick of our approach relies on the ability of a routing tree to compute scalar products as data flows from the leaf nodes to the base station. The computation can be readily implemented in systems such as TAG [56, 58], implemented in TinyOS, in which the routing tree is already designed to minimize energy consumption. The computation of scalar products within the network makes possible the implementation of basis changes in a distributed manner.

The PCA provides a basis change that brings several benefits. First, it removes correlations between data. Therefore the technique can be used to remove in a fully distributed way the spatial correlations in sensor measurements. Second, the PCA orders the basis vectors in such a way that most of the measurements can be represented by the first principal components. The PCA therefore allows to trade network load for accuracy, by simply changing the number of PC scores retrieved from the network. Third, thanks to this ability to order the basis vectors, the approach can be used to filter measurement noise, by discarding the PC scores of the minor principal components. Finally, the PCA is also a dimensionality reduction technique in the field of machine learning, which can be used to improve the accuracy of event detection or pattern recognition tasks for example.

Whenever data are correlated, which is often the case in sensor networks, the PCA allows to efficiently extract useful information. The combined use of the PCA with aggregation services such as TAG makes our approach particularly well adapted to sensor networks. We illustrated the ability the PCA to efficiently process sensor network data, by relying on two data sets. The first data set simulated a scenario where three different phenomena appeared in an environment monitored by 100 sensor nodes. We showed that the PCA was able to properly identify the three different phenomena and to recognize them with a high accuracy. The second data set consisted of real-world temperature measurements taken by 52 sensors in a large office. We showed that the PCA could represent most of the temperature variations with few principal components. For both data sets, the use of the DPCA could lead to significant savings in terms of network load.

7.2 Discussion and future research

Efficient data extraction from sensor networks is both a key issue and a challenging task. This section outlines future work that aims at making the methods presented in this thesis more robust, and at extending their uses to other applications.

Robustness to message loss : Wireless communication links are often prone to error, which can cause the loss of messages between sensor nodes. We assumed in this thesis that packet acknowledgments were used to ensure message delivery, which allowed us to make the convenient assumption that all messages sent were received by their intended recipient. The use of packet acknowledgements however causes network overhead, and they moreover do not guarantee the delivery of the message in case of permanent link failure for example.

It is in most cases important to detect when a message is lost, as it can have serious consequences on the reliability of the monitoring system [56, 61]. This is particularly true for approaches based on learning, which aim at reducing to a minimum the data collected. Each piece of data therefore usually has an important role in the overall system's reliability. For example, in aggregation approaches, the loss of one partial state record at node i makes the data from all sensor nodes in the subtree of sensor i lost.

The use of multi-path routing [70, 61] is a possibility to improve the robustness of message delivery. The increase in robustness is however obtained at the cost of an increased amount of radio communication. Moreover, multi-path routing makes the use of aggregation more difficult, as data sent over multiple paths may be aggregated more than once. Schemes like synopses [70] or Karumoto models are research tracks worth investigating for robust data aggregation. Finally, it is worth mentioning that if a loss is detected, learning techniques may for instance be used to estimate the value of the missing piece of information [80].

Erroneous measurements : Sensors are prone to failure of malfunctioning. Erroneous measurements can therefore appear in the flow of collected readings. For reasons similar to message loss, the detection of erroneous readings is an important issue. The detection of such data, also known as outliers, is however a non trivial task. We mention as solutions to this issue the possibility of adding statistical tests on the collected readings, as discussed in [63], or a priori bounds (e.g., all temperature readings outside the range [-20° C,60° C] should be flagged as erroneous).

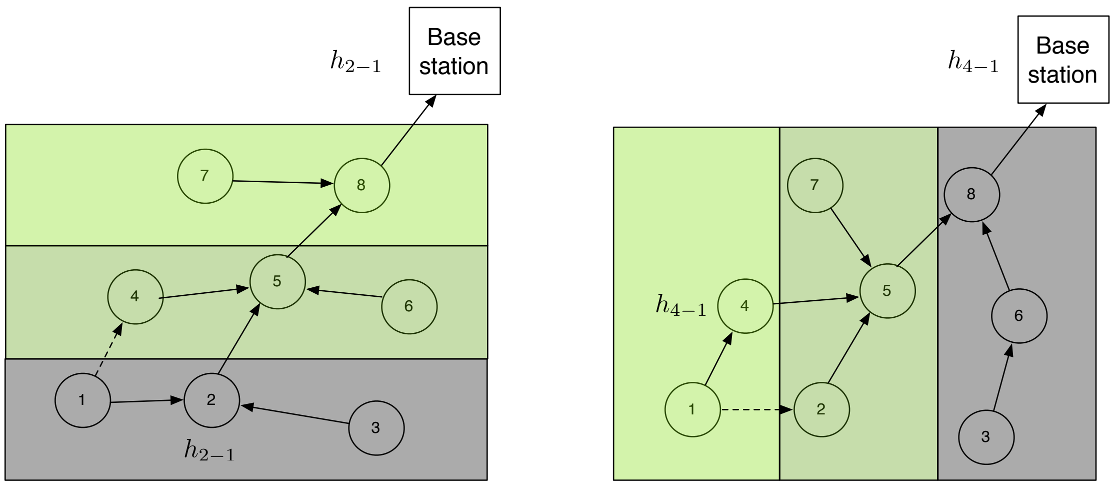


Figure 7.1: Shaded areas represent regions with high correlations. The ability of edge monitoring to provide good predictions is tightly coupled to the routing tree.

Cross-layering optimization: The problem of energy-efficient design in sensor networks has been addressed by various techniques at different layers of the system, ranging from the sensor hardware to the network routing to the application. The optimization within each individual layer however often leads to inefficient solutions [45]. To illustrate this point, let us consider the edge monitoring approach (Section 3.3), and the routing trees represented in Figure 7.1. Each shaded region corresponds to a region with high correlation. In the left configuration, it is preferable to monitor the edge $1-2$, while in the second case, it is preferable that edge $1-4$ is monitored. The edge monitoring can therefore be optimized by integrating in the routing choices information related to the monitoring application.

Joint optimization across layers is referred to as *cross-layering optimization* [25, 45]. In the DCPC, an interesting cross-layer optimization problem is brought by the local covariance hypothesis. On most sensor nodes, the radio power can be set at different levels in order to change the communication range of the sensor node. This feature could be used to set the radio power so that most of the covariances between neighboring nodes are captured.

A second cross-layer optimization is to reduce the largest number of children in the routing tree. The network load of the PCAg mainly depends on this parameter, and therefore its minimization can allow to further reduce the network load of the PCAg approach.

Non-stationarity: The techniques presented in this thesis mainly relied on the hypothesis that the joint probability distribution of the data was stationary over time. In the AMS, this hypothesis is actually used to motivate the use of the racing algorithm. In case of non-stationary data, different strategies must be considered. In particular, the racing is not appropriate, and the whole collection of models should remain in competition. A possible approach to deal with non-stationarity is to use *recursive learning approaches with forgetting factors* [37, 68]. Recursive learning allows to update the parameters of the prediction model as new observations become available, while the forgetting factor discards the contributions of older observations. For linear models such as autoregressive models or the principal component analysis, there exists such recursive formulations which could be used.

Multisensor fusion with the DPCA: Most sensor nodes have several sensors, allowing a sensor node to sense different physical quantities. These quantities are sometimes highly correlated (e.g. humidity and temperature [19, 15]), and the computation of the projection basis could be extended to include different physical quantities.

Multichannel Single Spectrum Analysis: Sensor data are very often correlated over time. The PCA can be extended to the time domain by integrating lagged measurements in the set of variables. The technique, called *Multichannel Single Spectrum Analysis* [42], allows to compress data both over space and time, but causes an increased latency in the delivery of the aggregates. It was briefly mentioned in Section 5.2, and deserves additional theoretical and experimental analysis.

Independent Component Analysis: The aggregation principle underlying the PCAg is also readily extensible to any basis transformation. Among the basis transformations of interest, we stress that the independant component analysis (ICA), also known as blind source separation [37, 16], is particularly appealing. ICA aims at determining a basis which not only decorrelates signals, but that also gets them independent. ICA has for example proven particularly efficient in speech processing in separating the set of independent sources composing an audio signal.

Bounded approximation with the PCAg: This approach, outlined in Section 5.2, is particularly interesting as it allows to ensure the observer that approximations are ϵ-bounded. The preliminary analysis (Section 5.3) of the tradeoffs between the error tolerance ϵ, the number of PCs q, and the highest network load could be further investigated. In particular, an interesting issue would be to determine, for a given ϵ, what is the parameter q which provides the lowest HNL. Also, for a given q, there probably exists a relationship between the proportion of retained variance and the error threshold ϵ. The identification a such a relationship would allow to estimate the HNL directly as a function of q. Further analytical and experimental work is needed in this direction.

Combining aggregation and adaptive model selection: The AMS can be run between children and parent nodes in the routing tree, with an error threshold ϵ_{AMS}, and the PCAg can rely on the AMS to aggregate data. Combining aggregation and adaptive model selection however increases the approximation errors entailed by the AMS. The resulting aggregate, for a network of S sensors, would be within $\pm S\epsilon_{\text{AMS}}$ of the true projections. The approximation error therefore grows linearly with the network size. It is therefore not clear whether this combination can be of practical interest. Further experimental work should be carried out to investigate this question.

Implementation of the DPCA on wireless sensors : The TinyOS [54] operating system has become a reference for the programming of sensor nodes. Though many projects have already developed their own codes for allowing aggregation [56, 27, 13], these implementations were either developed for ad-hoc tests, or made proprietary (such as TAG). We believe that an open source library of functions would be of interest in the dissemination of these ideas.

7.3 Learning with invisible media

Wireless sensor networks are the latest trend of technological advances which were well predicted by Gordon Moore, the co-founder of Intel, some 45 years ago [69]. Known as the *Moore's law* in the field of computer science, the law states that the number of transistors that can be placed inexpensively on an integrated circuit doubles every two years. This law has been remarkably verified since it was stated, and computing devices have become more and more part of our daily lives. Figure 7.2 illustrates the evolution of computing devices, starting from the Eniac.

The Eniac was created in 1946, at the time referred to as the "giant brain", and had a volume of 63m^2 and consumed 150kW of power [34]. Advances in electronics have allowed to reduce the price and volume of computational devices, which made their use possible to a wider audience. The 90s definitely made a corner in the history of computer science, by allowing many people to have their own *Personal Computer*, which could be used as a new communication device thanks to *Internet.* Cell phones have since then allowed to make communication possible anywhere and anytime, both using cellular networks and the Internet.

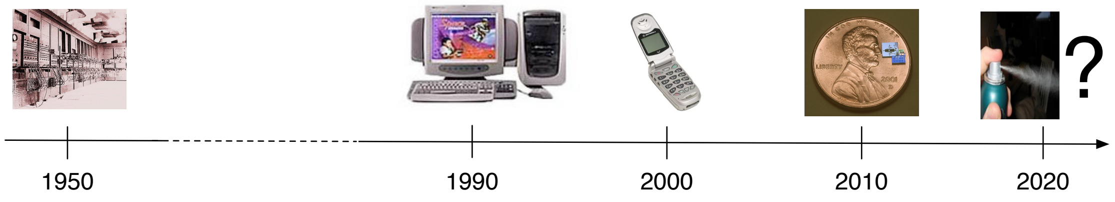


Figure 7.2: Moore's law: computing devices become smaller and cheaper over time.

Wireless sensor networks nowadays represent what will probably be tomorrow's part of our daily lives. WSNs allow to extend Internet to the physical world, providing real-time information about physical phenomena occurring in an environment. This thesis brought additional results to the feasibility of such real-time monitoring scenarios. In particular, we showed that learning techniques could be efficiently used to reduce the communication among sensor nodes.

The challenges brought by these new technologies are still numerous, and we believe that intelligent data processing techniques can address some of the most important. Data is at the heart of these sensing systems, and is what actually matters to the observer. The exponential amounts of data that such systems will generate in the coming years cannot however be made readily visible to the observer. Rather, data fusion and learning techniques must be further developed to allow collaborative and intelligent processing of the data *within* the network.

Appendix A

Statistical Framework

This appendix presents a brief review of the statistical principles and terminology encountered in this thesis. In particular, we define the notions of random variables, random vectors, probability density functions and stochastic processes.

Random experiments

A sensor is an electronic device whose accuracy depend on a number of factors that are rarely fully controlled. As such, a measurement collected by a sensor does not reflect exactly the physical quantity monitored, but rather an approximated value. Depending on the quality of the sensing device and on the care taken in controlling the conditions of the environment, the accuracy of the measurements may be improved. In any case, a number of factors influence the eventual measurement in such a way that, if it was possible to take two measurements at the same time and location, these would inevitably differ. A measure obtained by a sensor at one point of space and time is, in statistical terms, the result of a *random experiment.*

Definition A.1: *A* ***random experiment*** *is characterized by three elements:*

1. *A set Ξ of possible* outcomes *ξ of the experiment.*
2. *A set Ω of subsets $\omega \subset \Xi$. Each subset is called an* event. *Ω must be a Borel field, i.e., a nonempty class of sets such that*

$$\textit{if } \omega \in \Omega \textit{ then } \bar{\omega} \in \Omega \tag{A.1}$$

$$\textit{if } \omega_i \in \Omega, i = 1, \ldots, l, \textit{ then } \cup_{i=1}^{l} \omega_i \in \Omega \tag{A.2}$$

   where $\bar{\omega}$ is the complementary of ω and the number l of events can be finite of infinite. It follows that

$$\{\emptyset\} \in \Omega \textit{ and } \Xi \in \Omega \tag{A.3}$$

   where $\emptyset$ is the empty set, describing the event that never occurs.

3. *A* probability measure *$\mathcal{P} : \Omega \rightarrow [0, 1]$ defined on the elements ω. The function $\mathcal{P}$ must satisfy the following three conditions:*

$$\begin{aligned} \mathcal{P}(\omega) &\geq 0 && \text{(A.4)} \\ \mathcal{P}(\Xi) &= 1 && \text{(A.5)} \\ \textit{if } \omega_1 \cap \omega_2 = \{\emptyset\} \textit{ then } \mathcal{P}(\omega_1 \cap \omega_2) &= \mathcal{P}(\omega_1) + \mathcal{P}(\omega_2) && \text{(A.6)} \end{aligned}$$

The triple $(\Xi, \Omega, \mathcal{P})$ *is also called the* probabilistic model *of an experiment.*

Probabilistic model

This definition of a random experiment introduces the important notion of a probabilistic model, which allows to represent the uncertainty associated with a random experiment. The random experiments encountered in this thesis always deal with outcomes that are naturally associated with real numbers, such as the measurement of a physical quantity by a sensor. This mapping between the space of outcomes and the set of real numbers is formalized by the notion of *real random variable.*

Definition A.2: *A* ***real random variable*** $\mathbf{x}$ *is a real function whose domain is the space* Ξ*. It assigns a real number* $\mathbf{x}(\xi)$ *to every outcome* ξ *of the random experiment. The set* $\{\mathbf{x} \leq x\}$ *denotes the set of outcomes* ξ *such that* $\mathbf{s}(\xi)$ *is an event. A random variable verifies the two following conditions:*

1. *The set* $\{\mathbf{x} \leq x\}$ *is an event for any real number* s*.*
2. *The probability of the events* $\{\mathbf{x} = +\infty\}$ *and* $\{\mathbf{x} = -\infty\}$ *equals 0.*

Unless otherwise stated, it will be assumed that all random variables (r.v.) are continuous and real. Random variables allow to quantify the probabilities of events by means of the cumulative distribution function and the probability density function.

Definition A.3: *The* ***cumulative distribution function*** *(cdf)* $F_{\mathbf{x}}(x)$ *is the function*

$$F_{\mathbf{x}}(x) = \mathcal{P}(\{\mathbf{x} \leq x\})$$

defined for any number $x \in \mathbb{R}$*.*

Thus, for a given x, $F_{\mathbf{x}}(x)$ equals the probability of the event $\{\mathbf{x} \leq x\}$ such that $\mathbf{x}(\xi) < x$. For brevity, we shall also say that $F_{\mathbf{x}}(x)$ equals the probability that $\mathbf{x} \leq x$. For continuous random variables, the cdf is a nonnegative, nondecreasing continuous function whose values lie in the interval $0 \leq F_{\mathbf{x}}(x) \leq 1$. From the above definition, it follows that $F_{\mathbf{x}}(-\infty) = 0$ and $F_{\mathbf{x}}(+\infty) = 1$.

Definition A.4: *The* ***probability density function*** *(pdf)* $p_{\mathbf{x}}(x)$ *of a continuous random variable* $\mathbf{x}$ *is the derivative of the cumulative distribution function:*

$$p_{\mathbf{x}}(x) = \left.\frac{F_{\mathbf{x}}(\mathbf{x})}{d\mathbf{x}}\right|_{\mathbf{x}=x}$$

For the sake of simplicity, $F_{\mathbf{x}}(x)$ and $p_{\mathbf{x}}(x)$ shall be denoted by $F(x)$ and $p(x)$, respectively. The subscript referring to the random variable shall be used only if confusion is possible.

Example A.1: *The Gaussian (or normal) probability distribution is used in numerous models and applications, and is encountered at several occasions throughout this thesis. Its pdf is given by*

$$p(x) = \frac{1}{\sqrt{2\pi\sigma_{\mathbf{x}}}} \exp\left(-\frac{(x-\mu_{\mathbf{x}})^2}{2\sigma_{\mathbf{x}}}\right)$$

where $\mu_{\mathbf{x}}$ *and* $\sigma_{\mathbf{x}}$ *are known as the mean and variance of the random variable* $\mathbf{x}$*.*

Definition A.5: *A* ***real random vector*** $\mathbf{x}$ *is a* $n-$*dimensional vector*

$$\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)^T$$

where the elements $\mathbf{x}_1$*,* $\mathbf{x}_2$*,* $\ldots$*,* $\mathbf{x}_n$ *are random variables. The superscript* T *denotes the transpose, as all vectors in this thesis shall be by convention column vectors. The index* (j) *shall refer to elements of a vector.*

The concepts of probability distribution generalizes easily to random vectors, and the definitions above applies exactly in the manner, replacing variables x and random variables $\mathbf{x}$ with their vector counterparts. No font distinction is made between vectors and scalars.

Example A.2: *The pdf of a multivariate Gaussian probability distribution is given by*

$$p(\mathbf{x} = x) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} \exp^{\left(-\frac{1}{2}(x-\mu_{\mathbf{x}})^T \Sigma^{-1} (x-\mu_{\mathbf{x}})\right)}$$

where $\mu_{\mathbf{x}}$ *and* Σ *are the mean and covariance matrix of the random vector* $\mathbf{x}$*.*

Stochastic processes

Sensing devices are however subject to some measurement noise, and therefore the set of measurements collected by sensors over space and/or time is the result of random experiments. In the case of spatiotemporal data, the randomness associated to a scalar field is represented in statistics by *stochastic processes*, which are families or collections of random variables located and/or indexed by spatial and/or time metric. In the most general definition,

Definition A.6: *Given a probabilistic model* $(\Xi, \Omega, \mathcal{P})$*, a* ***spatiotemporal stochastic process*** *is a family of real random variables* $\mathbf{x}(c, t, \xi)$

$$\{\mathbf{x}(c, t, \xi) : c \in \ \mathbb{R}^d, t \in \mathbb{R}^*, \xi \in \Xi\}$$

where the inputs $c \in \mathbb{R}^d$ *refers to coordinates in space,* $t \in \mathbb{R}^*$ *refers to the time domain, and* $\xi \in \Xi$ *refers to the space of possible outcomes.*

When a stochastic process is restricted to the time domain, it is referred to as a *time series.*

Definition A.7: *Given a probabilistic model* $(\Xi, \Omega, \mathcal{P})$*, a* ***time stochastic process****, or* ***time series****, is a collection of real random variables* $\mathbf{x}(t, \xi)$ *indexed by a time domain* $\mathcal{T}$

$$\{\mathbf{x}(t, \xi) : t \in \mathcal{T}\}$$

A particular case, which serve in this thesis the purpose of modeling streams of sensor data, is when the time domain is defined as $\mathcal{T} = \mathbb{N}^*$. Such a process is called discrete-time stochastic process, or discrete time series, and will be denoted $\mathbf{x}[t]$. The random nature of $\mathbf{x}[t]$ shall be implicitly assumed by the use of the bold font. The notation $x[t]$ will be used to represent a particular realization of a discrete-time stochastic process $\mathbf{x}[t]$.

Definition A.8: *A process is called a random process if it consists of a sequence of random variables* $\mathbf{x}[t]$ *which are mutually independent and identically distributed*

The notation $\boldsymbol{\nu}[t]$ will be used to refer to a discrete-time random process.

Appendix B

Recursive computation of the covariances

This Appendix gives the details of the derivations underlying Equations (5.12) and (5.13). Equation (5.12) states that

$$\begin{aligned}
\hat{\Sigma}[N] &= \frac{1}{N-1}\sum_{t=1}^{N}(s[t]-\hat{\mu}[N])(s[t]-\hat{\mu}[N])^T \\
&= \frac{1}{N-1}X[N]^T X[N] - \frac{N}{N-1}\hat{\mu}[N]\hat{\mu}[N]^T
\end{aligned}$$

where $\hat{\mu}[N] = \frac{1}{N}\sum_{t=1}^{N} s[t]$ is an estimate of the average vector of observations $s[t]$. The result is obtained by developing the product

$$\begin{aligned}
\hat{\Sigma}[N] &= \frac{1}{N-1}\sum_{t=1}^{N}(s[t]-\hat{\mu}[N])(s[t]-\hat{\mu}[N])^T \\
&= \frac{1}{N-1}\left(\sum_{t=1}^{N}(s[t]s[t]^T) - 2\sum_{t=1}^{N}\hat{\mu}[N]s[t]^T + \sum_{t=1}^{N}\hat{\mu}[N]\hat{\mu}[N]^T\right) \\
&= \frac{1}{N-1}\left(\sum_{t=1}^{N}(s[t]s[t]^T) - 2\hat{\mu}[N]\sum_{t=1}^{N}s[t]^T + N\hat{\mu}[N]\hat{\mu}[N]^T\right) \\
&= \frac{1}{N-1}\left(\sum_{t=1}^{N}(s[t]s[t]^T) - 2N\hat{\mu}[N]\hat{\mu}[N]^T + N\hat{\mu}[N]\hat{\mu}[N]^T\right) \\
&= \frac{1}{N-1}\sum_{t=1}^{N}X[N]^T X[N] - \frac{N}{N-1}\hat{\mu}[N]\hat{\mu}[N]^T.
\end{aligned}$$

which gives the simplified formulation. Therefore, each element $\hat{\sigma}_{(i,j)}[N]$ of $\hat{\Sigma}[N]$ can be expressed as

$$\hat{\sigma}_{(i,j)}[N] = \frac{1}{N-1}\sum_{t=1}^{N}s_i[t]s_j[t] - \frac{N}{N-1}\hat{\mu}_i[N]\hat{\mu}_j[N].$$

Since $\hat{\mu}_i[N] = \frac{1}{N}\sum_{t=1}^{N} s_i[t]$, we have

$$\begin{aligned}\hat{\sigma}_{(i,j)}[N] &= \frac{1}{N-1}\sum_{t=1}^{N} s_i[t]s_j[t] - \frac{N}{N^2(N-1)}\sum_{t=1}^{N} s_i[t]\sum_{t=1}^{N} s_j[t] \\ &= \frac{1}{N-1}\sum_{t=1}^{N} s_i[t]s_j[t] - \frac{1}{N(N-1)}\sum_{t=1}^{N} s_i[t]\sum_{t=1}^{N} s_j[t].\end{aligned}$$

Denoting by

$$r_i[N] = \sum_{t=1}^{N} s_i[t]$$

and

$$r_{(i,j)}[N] = \sum_{t=1}^{N} s_i[t]s_j[t]$$

we have

$$\hat{\sigma}_{(i,j)}[N] = \frac{1}{N-1} r_{(i,j)}[N] - \frac{1}{N(N-1)} r_i[N] r_j[N]$$

with

$$\begin{aligned} r_i[N] &= r_i[N-1] + s_i[N] \\ r_{(i,j)}[N] &= r_{(i,j)}[N-1] + s_i[N]s_j[N]\end{aligned}$$

from which Equation (5.13) is derived.

Appendix C

Convergence of the power iteration method

This Appendix presents the convergence properties [28] of the Power Iteration Method (the PIM). Denoting by Σ a covariance matrix $n \times n$, the idea of the PIM is to choose an initial n-element vector v_0, and to form the sequence

$$\begin{array}{rcccl} v_1 & = & \Sigma v_0 & & \\ v_2 & = & \Sigma v_1 & = & \Sigma^2 v_0 \\ & \vdots & & \vdots & \\ v_t & = & \Sigma v_{t-1} & = & \Sigma^t v_0 \\ & \vdots & & \vdots & \end{array}$$

Let w_k, $1 \leq k \leq n$ be the eigenvectors of Σ. The eigenvectors form a basis for the n-dimensional space, and therefore we can write, for arbitrary v_0,

$$v_0 = \sum_{k=1}^{n} \theta_k w_k$$

where θ_k are some set of coefficients. Then,

$$v_1 = \Sigma v_0 = \sum_{k=1}^{n} \theta_k \Sigma w_k = \sum_{k=1}^{n} \theta_k \lambda_k w_k$$

where λ_k, $1 \leq k \leq n$ are the eigenvalues of Σ. Continuing, we get for $t = 2, 3, \ldots$

$$v_t = \sum_{k=1}^{n} \theta_k \lambda_k^t w_k$$

and it follows

$$\frac{v_t}{\theta_1 \lambda_1^t} = \left(w_1 + \frac{\theta_2}{\theta_1} \left(\frac{\lambda_2}{\lambda_1} \right)^t w_2 + \ldots + \frac{\theta_n}{\theta_1} \left(\frac{\lambda_n}{\lambda_1} \right)^t w_n \right).$$

Assuming that the first eigenvalue of Σ is distinct from the remaining eigenvalues, i.e., $\lambda_1 > \lambda_2 \geq \ldots \geq \lambda_n$, it follows that

1. a normalized version of $v_t \rightarrow w_1$ as $t \rightarrow \infty$, and
2. the ratios of the corresponding elements of v_t and $v_{t-1} \rightarrow \lambda_1$ ad $t \rightarrow \infty$.

The speed of convergence of the PIM depends mainly on ratio $\frac{\lambda_2}{\lambda_1}$. The higher the ratio, the faster the convergence. The speed of convergence also depends on the choice of the initial vector v_0, and the convergence is more rapid if v_0 is close to w_1.

The PIM is therefore slow for computing the main eigenvector of a matrix when λ_2 is close to λ_1. In the case of equality, such that $\lambda_1 = \lambda_2 > \lambda_3$, a similar development as above shows that a normalized version of $v_t \rightarrow w_1 + \frac{\theta_2}{\theta_1} w_2$ as $t \rightarrow \infty$, and therefore does not convergence to w_1 anymore.

Although this is a limit of the PIM in problems where exact eigenvectors are sought, it is for the problem considered in this thesis not such an issue. The eigenvectors are required for the PCA, and used as they are the vectors that best represent the data. The amount of data retained by the k-th eigenvector, in terms of variance, is the k-th eigenvalue. In case of equality between two eigenvalues, then it means for the PCA that the corresponding eigenvectors have the same efficiency at representing the data. This motivates that the maximum number of iterations $t_{\max}$ in Algorithm 6 can be set low.

Appendix D

Additional Results for the DCPC

Image data set - QR and PIM NMSE for SNR5 and SNR05.

$t_{\max} = 50$					
	1 PC	2 PCs	3 PCs	4 PCs	5 PCs
QR NMSE	0.713	0.448	0.240	0.238	0.235
PM NMSE	0.713	0.448	0.240	0.238	0.235
Time conv	43.200	34.300	3.000	49.900	46.800
$t_{\max} = 10$					
	1 PC	2 PCs	3 PCs	4 PCs	5 PCs
QR NMSE	0.713	0.448	0.240	0.238	0.235
PM NMSE	0.716	0.482	0.240	0.238	0.235
Time conv	10.000	10.000	3.000	10.000	10.000

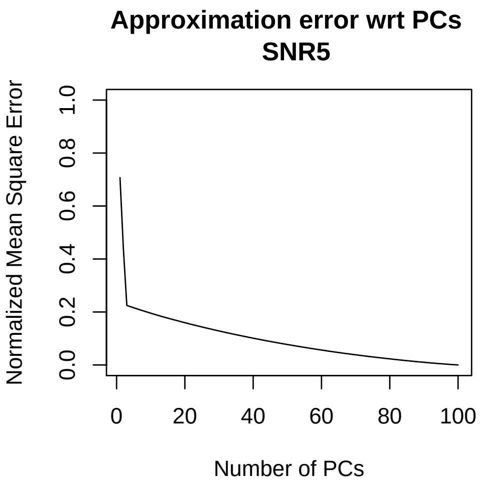


Figure D.1: Comparison of the QR and PIM methods in terms of NMSE. The table (left) gives the NMSE for one up to five PCs, and the convergence times for the PIM method. The right figure gives a visual plot of the NMSE for the QR method. The data set is SNR5.

$t_{\max} = 50$					
	1 PC	2 PCs	3 PCs	4 PCs	5 PCs
QR NMSE	0.913	0.827	0.758	0.750	0.742
PM NMSE	0.914	0.829	0.758	0.750	0.742
Time conv	37.500	36.000	6.800	48.800	46.800
$t_{\max} = 10$					
	1 PC	2 PCs	3 PCs	4 PCs	5 PCs
QR NMSE	0.913	0.827	0.758	0.750	0.742
PM NMSE	0.916	0.840	0.758	0.750	0.742
Time conv	10.000	10.000	4.700	10.000	10.000

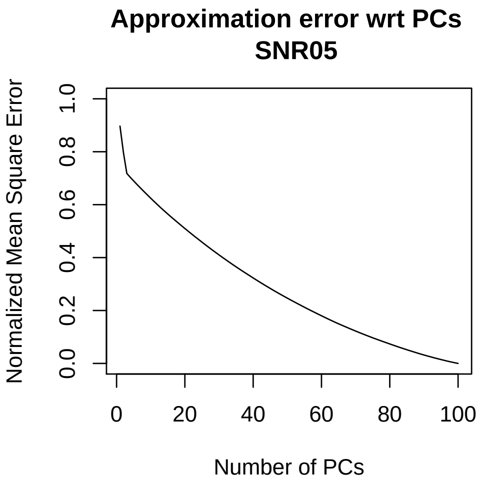
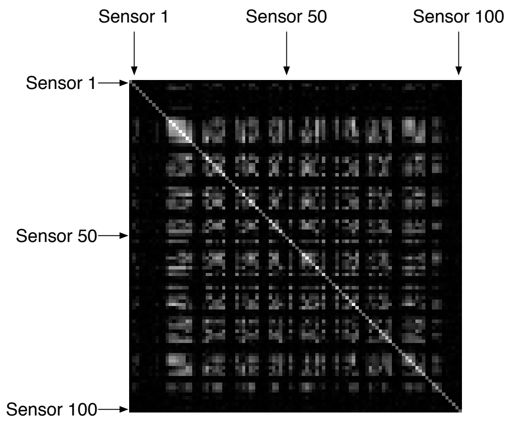
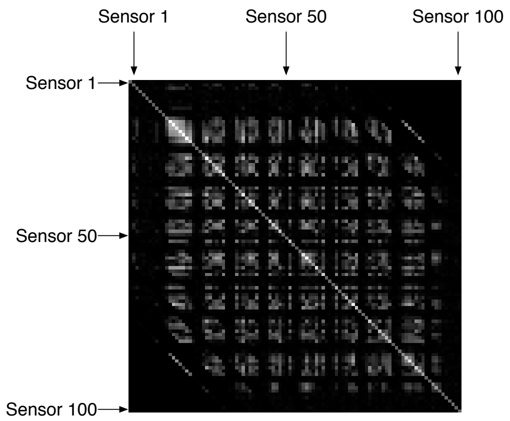


Figure D.2: Comparison of the QR and PIM methods in terms of NMSE. The table (left) gives the NMSE for one up to five PCs, and the convergence times for the PIM method. The right figure gives a visual plot of the NMSE for the QR method. The data set is SNR05.

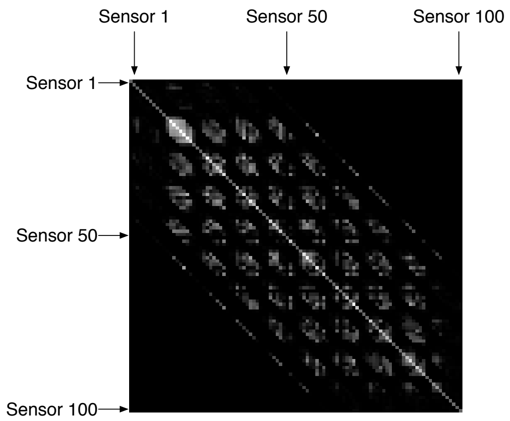
Image data set - SNR1 - Visual representation of the local covariance hypothesis.



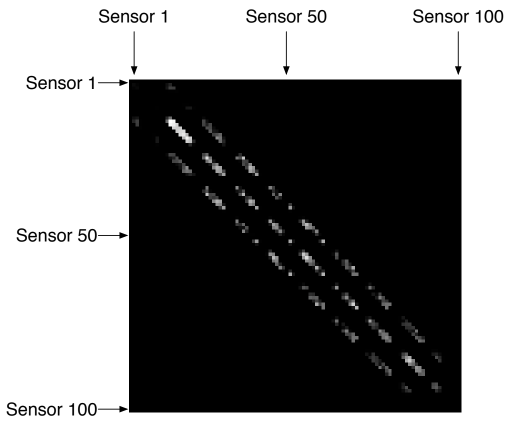
(a)



(b)



(c)



(d)

Figure D.3: Covariances matrices resulting from the local covariance hypothesis. The covariances are normalized between 0 and 1 and represented by gray levels. (a) Full covariance matrix. (b) Radio range 70 meters. (c) Radio range 40 meters. (d) Radio range 15 meters.

Intel data set - Convergence time for incomplete covariance matrices.

	1 PC	2 PCs	3 PCs	4 PCs	5 PCs	6 PCs	8 PCs	10 PCs	12 PCs	15 PCs
QR NMSE all	0.215	0.140	0.114	0.098	0.088	0.083	0.066	0.052	0.045	0.036
QR NMSE	0.215	0.140	0.114	0.098	0.088	0.083	0.066	0.052	0.045	0.036
PM NMSE	0.215	0.140	0.114	0.099	0.088	0.082	0.063	0.053	0.045	0.036
Time conv	2.700	4.900	7.400	8.700	9.800	9.800	10.000	9.900	10.000	10.000

Table D.1: Radio range 50 meters.

	1 PC	2 PCs	3 PCs	4 PCs	5 PCs	6 PCs	8 PCs	10 PCs	12 PCs	15 PCs
QR NMSE all	0.215	0.140	0.114	0.098	0.088	0.083	0.066	0.052	0.045	0.036
QR NMSE	0.379	0.241	0.138	0.127	0.103	0.090	0.082	0.075	0.070	0.063
PM NMSE	0.379	0.212	0.126	0.121	0.106	0.102	0.093	0.084	0.079	0.072
Time conv	6.800	9.800	9.400	10.000	10.000	10.000	10.000	10.000	10.000	10.000

Table D.2: Radio range 20 meters.

	1 PC	2 PCs	3 PCs	4 PCs	5 PCs	6 PCs	8 PCs	10 PCs	12 PCs	15 PCs
QR NMSE all	0.215	0.140	0.114	0.098	0.088	0.083	0.066	0.052	0.045	0.036
QR NMSE	0.547	0.510	0.380	0.299	0.213	0.192	0.111	0.079	0.067	0.055
PM NMSE	0.547	0.384	0.289	0.193	0.146	0.127	0.116	0.101	0.088	0.077
Time conv	7.800	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000

Table D.3: Radio range 10 meters.

	1 PC	2 PCs	3 PCs	4 PCs	5 PCs	6 PCs	8 PCs	10 PCs	12 PCs	15 PCs
QR NMSE all	0.215	0.140	0.114	0.098	0.088	0.083	0.066	0.052	0.045	0.036
QR NMSE	0.687	0.624	0.579	0.494	0.458	0.366	0.292	0.232	0.174	0.108
PM NMSE	0.671	0.579	0.491	0.352	0.285	0.245	0.191	0.161	0.132	0.110
Time conv	9.900	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000

Table D.4: Radio range 6 meters.

Appendix E

TinyOS - Interfaces ***Timer***, ***ADC*** and ***SendMsg***

This Appendix provides examples of the *Timer*, *ADC* and *SendMsg* TinyOS interfaces, taken from the distribution TinyOS-1.x [85].

Licensing

Below are the licensing conditions for the TinyOS files included in this Appendix.

Interface Timer.nc

```
/**
```

```
 * This interface provides a generic timer that can be used to generate
 * events at regular intervals.
 *
 * @author Su Ping
 * @author Sam Madden
 * @author David Gay
 * @modified 7/16/02
 */
includes Timer; // make TIMER_x constants available
interface Timer {

  /**
   * Start the timer.
   * @param type The type of timer to start. Valid values include
   *  'TIMER_REPEAT' for a timer that fires repeatedly, or
   *  'TIMER_ONE_SHOT' for a timer that fires once.
   *  @param interval The timer interval in <b>binary milliseconds</b> (1/1024
   *  second). Note that the
   *    timer cannot support an arbitrary range of intervals.
   *    (Unfortunately this interface does not specify the valid range
   *    of timer intervals, which are specific to a platform.)
   *  @return Returns SUCCESS if the timer could be started with the
   *    given type and interval. Returns FAIL if the type is not
   *    one of TIMER_REPEAT or TIMER_ONE_SHOT, if the timer rate is
   *    too high, or if there are too many timers currently active.
   */
  command result_t start(char type, uint32_t interval);

  /**
   * Stop the timer, preventing it from firing again.
   * If this is a TIMER_ONE_SHOT timer and it has not fired yet,
   * prevents it from firing.
   * @return SUCCESS if the timer could be stopped, or FAIL if the timer
   * is not running or the timer ID is out of range.
   */
  command result_t stop();

  /**
   * The signal generated by the timer when it fires.
   */
  event result_t fired();
}
```

Interface ADC.nc

```
includes ADC;
//includes sensorboard; // this defines the user names for the ports

/**
 * Analog to Digital Converter Interface. <p>
 * Defines the functions provided by any ADC
 *
 * @modified  6/25/02
 *
 * @author Jason Hill
 * @author David Gay
 * @author Philip Levis
 */
```

```
interface ADC {
  /**
   * Initiates an ADC conversion on a given port.
   *
   * @return SUCCESS if the ADC is free and available to accept the request
   */
  async command result_t getData();
  /**
   * Initiates a series of ADC conversions. Each return from
   * <code>dataReady()</code> initiates the next conversion.
   *
   * @return SUCCESS if the ADC is free and available to accept the request
   */
  async command result_t getContinuousData();

  /**
   * Indicates a sample has been recorded by the ADC as the result
   * of a <code>getData()</code> command.
   *
   * @param data a 2 byte unsigned data value sampled by the ADC.
   *
   * @return SUCCESS if ready for the next conversion in continuous mode.
   * if not in continuous mode, the return code is ignored.
   */
  async event result_t dataReady(uint16_t data);
}
```

Interface SendMsg.nc

```
includes AM;

/**
 * Basic interface for sending AM messages. Interface to the basic
 * TinyOS communication primitive.
 *
 * @author Jason Hill
 * @author David Gay
 * @author Philip Levis
 * @modified 6/25/02
 *
 *
 */

interface SendMsg
{
  command result_t send(uint16_t address, uint8_t length, TOS_MsgPtr msg);
  event result_t sendDone(TOS_MsgPtr msg, result_t success);
}
```

Bibliography

[1] D.W. Aha. *Lazy learning.* Kluwer Academic Publishers Norwell, MA, USA, 1997.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless Sensor Networks: A Survey. *Computer Networks*, 38(4):393–422, 2002.

[3] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[4] ST Alexander. *Adaptive Signal Processing: Theory and Applications.* Springer-Verlag New York, Inc. New York, NY, USA, 1986.

[5] Z. Bai et al. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide.* Society for Industrial and Applied Mathematics, 2000.

[6] M. Birattari, G. Bontempi, and H. Bersini. Lazy learning meets the recursive least squares algorithm. *Advances in Neural Information Processing Systems*, pages 375–381, 1999.

[7] C.M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, USA, 1995.

[8] G. Bontempi. Local learning techniques for modeling prediction and control. *Bruxelles: IRIDIA-Universite Libre de Bruxelles*, 1999.

[9] G. Bontempi and Y. Le Borgne. An adaptive modular approach to the mining of sensor network data. In *Proceedings of the Workshop on Data Mining in Sensor Networks, SIAM SDM*, pages 3–9, Philadeplhia, PA, 2005. SIAM Press.

[10] L. Breiman. *Classification and Regression Trees.* Chapman & Hall/CRC, 1998.

[11] P.J. Brockwell and R.A. Davis. *Introduction to time series and forecasting.* Springer, 2002.

[12] P. Buonadonna, D. Gay, J. M. Hellerstein, W. Hong, and S. Madden. TASK: Sensor Network in a Box. In *Proceedings of the 2nd IEEE European Workshop on Wireless Sensor Networks and Applications (EWSN'05)*, Istanbul, Turkey, February 2005.

[13] N. Burri and R. Wattenhofer. Dozer: ultra-low power data gathering in sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 450–459. ACM Press, 2007.

[14] C. Chatfield. *Time-series forecasting.* Chapman & Hall/CRC, 2001.

[15] D. Chu, A. Deshpande, J.M. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. In *International Conference on Data Engineering (ICDE)*, 2006.

[16] A. Cichocki and S. Amari. *Adaptive blind signal and image processing.* Wiley New York, 2002.

[17] N. Cristianini and J. Shawe-Taylor. *An introduction to support Vector Machines: and other kernel-based learning methods.* Cambridge University Press New York, NY, USA, 1999.

[18] Intel Research Laboratory Data. Project Website. berkeley.intel-research.net/labdata/, 2003.

[19] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-Driven Data Acquisition in Sensor Networks. In *Proceedings of the 30th Very Large Data Base Conference (VLDB'04)*, Toronto, Canada, 2004.

[20] A. Deshpande, C. Guestrin, S.R. Madden, J.M. Hellerstein, and W. Hong. Model-based approximate querying in sensor networks. *The VLDB Journal The International Journal on Very Large Data Bases*, 14(4):417–443, 2005.

[21] J.M. Dricot, M. Van Der Haegen, Y. Le Borgne, and G. Bontempi. A modular framework for user localization and tracking using machine learning techniques in wireless sensor networks. In *Proceedings of the 8th IEEE Conference on Sensors*, pages 1088–1091, 2008.

[22] J.M. Dricot, M. Van Der Haegen, Y. Le Borgne, and G. Bontempi. Performance evaluation of machine learning technique for the localization of users in wireless sensor networks. In L. Wehenkel, P. Geurts, and R. Marée, editors, *Proceedings of the BENELEARN Machine Learning Conference*, pages 93–94, 2008.

[23] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern classification.* Wiley New York, 2001.

[24] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, volume 4, 2001.

[25] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi. In-Network Aggregation Techniques for Wireless Sensor Networks: A Survey. *Wireless communications*, 14(2):70, 2007.

[26] D. Gay, P. Levis, R. Von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. *Sigplan Notices*, 38(5):1–11, 2003.

[27] J. Gehrke and S. Madden. Query processing in sensor networks. *Pervasive Computing, IEEE*, 3(1):46–55, 2004.

[28] G.H. Golub and C.F. Van Loan. *Matrix computations.* Johns Hopkins University Press, 1996.

[29] GoodFood EU Integrated Project: Food Safety and Quality Monitoring with Microsystems. Sensor Network in a Vineyard. www3.unifi.it/midra/goodfood/, 2007.

[30] C. Guestrin, P. Bodi, R. Thibau, M. Paski, and S. Madde. Distributed regression: an efficient framework for modeling sensor network data. *Proceedings of the third international symposium on Information processing in sensor networks*, pages 1–10, 2004.

[31] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[32] T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, J. Friedman, and R. Tibshirani. *The elements of statistical learning.* Springer New York, 2001.

[33] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 146–159. ACM Press, 2001.

[34] Eniac history. ENIAC. http://en.wikipedia.org/wiki/ENIAC.

[35] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, pages 13–30, 1963.

[36] L. Huang, X. Nguyen, M. Garofalakis, M. Jordan, A. Joseph, and N. Taft. In-network PCA and anomaly detection. In J. Platt B. Scholkopf and T. Hoffman, editors, *Proceedings of the 19th conference on Advances in Neural Information Processing Systems.* MIT Press, 2006.

[37] A. Hyvarinen, J. Karhunen, and E. Oja. *Independent Component Analysis.* J. Wiley New York, 2001.

[38] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 56–67, 2000.

[39] Intel Lab Data webpage. `http://db.csail.mit.edu/labdata/labdata.html`.

[40] A. Jain and E.Y. Chang. Adaptive sampling for sensor networks. *ACM International Conference Proceeding Series*, pages 10–16, 2004.

[41] Ankur Jain, Edward Y. Chang, and Yuan-Fang Wang. Adaptive Stream Resource Management Using Kalman Filters. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '04)*, pages 11–22, 2004.

[42] I.T. Jolliffe. *Principal Component Analysis.* Springer, 2002.

[43] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[44] G. Kerschen, P.D. Boe, J.C. Golinval, and K. Worden. Sensor validation using principal component analysis. *Smart Materials and Structures*, 14(1):36–42, 2005.

[45] B. Krishnamachari. *Networking Wireless Sensors.* Cambridge University Press New York, NY, USA, 2005.

[46] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 219–230. ACM Press, 2004.

[47] I. Lazaridis and S. Mehrotra. Capturing Sensor-Generated Time Series with Quality Guarantee. In *Proceedings of the 19th Intl. Conference on Data Engineering (ICDE'03)*, Bangalore, India, March 2003.

[48] Y. Le Borgne and G. Bontempi. Unsupervised and supervised compression with principal component analysis in wireless sensor networks. In *Proceedings of the Workshop on Knowledge Discovery from Data, 13th ACM International Conference on Knowledge Discovery and Data Mining*, pages 94–103, New York, NY, 2007. ACM Press.

[49] Y. Le Borgne, J.M. Dricot, and G. Bontempi. *Principal Component Aggregation for Energy-efficient Information Extraction in Wireless Sensor Networks*, chapter 5, pages 55–80. Taylor and Francis/CRC Press, 2008.

[50] Y. Le Borgne, M. Moussaid, and G. Bontempi. Simulation architecture for data processing algorithms in wireless sensor networks. In *Proceedings of the PAEWN workshop, 20th Conference on Advanced Information Networking and Applications*, pages 383–387, Piscataway, NJ, 2006. IEEE Press.

[51] Y. Le Borgne, S. Raybaud, and G. Bontempi. Distributed principal component analysis for wireless sensor networks. *Sensors Journal*, 8(8):4821–4850, August 2008.

[52] Y. Le Borgne, S. Santini, and G. Bontempi. Adaptive model selection for time series prediction in wireless sensor networks. *Journal of Signal Processing*, 87(12):3010–3020, December 2007.

[53] M. Lefebvre. *Applied stochastic processes Universitext*. Springer, 2007.

[54] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. TinyOS: An Operating System for Sensor Networks. *Ambient Intelligence*, pages 115–148, 2005.

[55] J. Li and Y. Zhang. Interactive sensor network data retrieval and management using principal components analysis transform. *Smart Materials and Structures*, 15:1747–1757(11), December 2006.

[56] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proceedings of the 5th ACM Symposium on Operating System Design and Implementation (OSDI)*, volume 36, pages 131 – 146. ACM Press, 2002.

[57] S.R. Madden. *The Design and Evaluation of a Query Processing Architecture for Sensor Networks*. PhD thesis, UNIVERSITY OF CALIFORNIA, 2003.

[58] S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–173, 2005.

[59] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, 2002.

[60] G. Manes, R. Fantacci, F. Chiti, M. Ciabatti, G. Collodi, D. Di Palma, and A. Manes. Enhanced System Design Solutions for Wireless Sensor Networks applied to Distributed Environmental Monitoring. In *Proceedings of the 32nd IEEE Conference on Local Computer Networks*, pages 807–814. IEEE Computer Society Washington, DC, USA, 2007.

[61] A. Manjhi, S. Nath, and P.B. Gibbons. Tributaries and deltas: Efficient and robust aggregation in sensor network streams. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 287–298. ACM New York, NY, USA, 2005.

[62] J. Mantyjarvi, J. Himberg, T. Seppanen, and N.R. Center. Recognizing human motion with multiple acceleration sensors. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, volume 2, 2001.

[63] M. Markou and S. Singh. Novelty detection: a review—part 1: statistical approaches. *Signal Processing*, 83(12):2481–2497, 2003.

[64] O. Maron. *Hoeffding Races: Model Selection for MRI Classification.* PhD thesis, Massachusetts Institute of Technology, 1994.

[65] O. Maron and A. W. Moore. The Racing Algorithm: Model Selection for Lazy Learners. *Artificial Intelligence Review*, 11(1-5):193–225, February 1997.

[66] K. Martinez, P. Padhy, A. Elsaify, G. Zou, A. Riddoch, JK Hart, and HLR Ong. Deploying a Sensor Network in an Extreme Environment. *IEEE SUTC*, 1:186–193, 2006.

[67] A. A. Miranda, Y. Le Borgne, and G. Bontempi. New routes from minimal approximation error to principal components. *Neural Processing Letters*, 27(3):197 – 207, June 2008.

[68] T.M. Mitchell. Machine Learning. *Mac Graw Hill*, 1997.

[69] GE Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.

[70] S. Nath, P.B. Gibbons, S. Seshan, and Z. Anderson. Synopsis diffusion for robust aggregation in sensor networks. 2008.

[71] National Oceanic Atmospheric Administration's National Data Buoy Center. www.ndbc.noaa.gov/historical_data.shtml, 2005.

[72] C. Olston, B.T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. *ACM SIGMOD Record*, 30(2):355–366, 2001.

[73] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling Ultra-Low Power Wireless Research. In *Proceedings of the 4th Intl. Conference on Information Processing in Sensor Networks: Special Track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN'05/SPOTS)*, April 2005.

[74] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, pages 364–369, 2005.

[75] J. Porter, P. Arzberger, H.W. Braun, P. Bryant, S. Gage, T. Hansen, P. Hansen, C.C. Lin, F.P. Lin, T. Kratz, et al. Wireless Sensor Networks for Ecology. *BioScience*, 55(7):561–572, 2005.

[76] R. Rajagopalan and P.K. Varshney. Data aggregation techniques in sensor networks: A survey. *IEEE Communications Surveys and Tutorials*, 8(4):48–63, 2006.

[77] P.J. Rousseeuw and G. Molenberghs. Transformation of non positive semidefinite correlation matrices. *Communications in Statistics-Theory and Methods*, 22(4):965–984, 1993.

[78] Silvia Santini. Research homepage. www.tinyos.net.

[79] Silvia Santini and Kay Römer. An adaptive strategy for quality-based data reduction in wireless sensor networks. In *Proceedings of the 3rd Intl. Conf. on Networked Sensing Systems (INSS 2006)*, Chicago, IL, USA, June 2006.

[80] A. Silberstein, R. Braynard, G. Filpus, G. Puggioni, A. Gelfand, K. Munagal, and J. Yang. Data-driven processing in sensor networks. *Ambient Intelligence*, pages 115–148, 2005.

[81] A. Stenman, F. Gustafsson, and L. Ljung. Just in Time Models for Dynamical Systems. In *Proceedings of the 35th IEEE Conference on Decision and Control*, volume 1, pages 1115–1120, Kobe, Japan, 1996.

[82] Ananthram Swami, Qing Zhao, Yao-Win Hong, and Lang Tong. *Wireless Sensor Networks: Signal Processing and Communications*. John Wiley & Sons, 2007.

[83] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. Habitat monitoring with sensor networks. *Communications of the ACM*, 47(6):34–40, 2004.

[84] S. Tilak, N.B. Abu-Ghazaleh, and W. Heinzelman. A taxonomy of wireless micro-sensor network models. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(2):28–36, 2002.

[85] TinyOS. Project Website: `http://www.tinyos.net`.

[86] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, et al. A macroscope in the redwoods. *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 51–63, 2005.

[87] D. Tulone and S. Madden. PAQ: Time Series Forecasting for Approximate Query Answering in Sensor Networks. In *Proceedings of the 3rd European Workshop on Wireless Sensor Networks*, pages 21–37. Springer, 2006.

[88] S. Wang, J. Yang, N. Chen, X. Chen, and Q. Zhang. Human Activity Recognition with User-Free Accelerometers in the Sensor Networks. In *International Conference on Neural Networks and Brain*, volume 2, 2005.

[89] X. Wang and H. Qi. Acoustic target classification using distributed sensor arrays. In *IEEE International conference on acoustics speech and signal processing*, volume 4, pages 4186–4186. IEEE; 1999, 2002.

[90] Brett Warneke, Matt Last, Brian Liebowitz, and Kristofer S. J. Pister. Smart dust: Communicating with a cubic-millimeter computer. *Computer*, 34(1):44–51, 2001.

[91] D.S. Watkins. Understanding the QR algorithm. *SIAM review*, pages 427–440, 1982.

[92] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees. Deploying a Wireless Sensor Network on an Active Volcano. *IEEE Internet Computing*, pages 18–25, 2006.

[93] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record*, 31(3):9–18, 2002.

[94] C. Zhang, M. Li, M. Wu, and W. Zhang. Monitoring Wireless Sensor Networks Using a Model-Aided Approach. *Lecture Notes in Computer Science*, 3976:1210, 2006.

[95] F. Zhao and L.J. Guibas. *Wireless Sensor Networks: An Information Processing Approach.* Morgan Kaufmann, 2004.