Time Series Prediction for Energy-Efficient Wireless Sensors: Applications to Environmental Monitoring and Video Games

Yann-Aël Le Borgne and Gianluca Bontempi

Machine Learning Group, Computer Science Department, CP212,
Faculty of Sciences, Université Libre de Bruxelles,
Bd Triomphe, Brussels, 1050, Belgium
{yleborgn,gbonte}@ulb.ac.be
http://mlg.ulb.ac.be

Abstract. Time series prediction techniques have been shown to significantly reduce the radio use and energy consumption of wireless sensor nodes performing periodic data collection tasks. In this paper, we propose an implementation of exponential smoothing, a standard time series prediction technique, for wireless sensors. We rely on a framework called *Adaptive Model Selection* (AMS), specifically designed for running time series prediction techniques on resource-constrained wireless sensors. We showcase our implementation with two demos, related to environmental monitoring and video games. The demos are implemented with TinyOS, a reference operating system for low-power embedded systems, and TMote Sky and TMote Invent wireless sensors.

Keywords: Wireless sensors, energy-efficiency, machine learning, time series prediction, exponential smoothing.

1 Introduction

Wireless sensor measurements typically follow temporal patterns, which are well approximated by time series prediction techniques. Different approaches have been proposed in the literature to approximate, by means of *parametric predictive models*, the measurements collected by wireless sensors [5, 9, 13, 15]. The rationale of these approaches is that, if the parametric predictive model follows the sensor's measurements with sufficient accuracy, then it is enough to communicate the parameters of the model instead of the real measurements. In [7], a generic framework called *Adaptive Model Selection* was proposed, which encompassed previously proposed approaches based on time series prediction for wireless sensors. AMS was shown to provide, for a wide range applications, significant communication and energy savings.

In this paper, we investigate the use of exponential smoothing techniques in the AMS framework. Exponential smoothing (ES) is a standard time series prediction technique, known to perform well in many real-world applications [12]. We implement ES in TinyOS [14], a reference operating system for low power

F. Martins, L. Lopes, and H. Paulino (Eds.): S-Cube 2012, LNICST 102, pp. 63–72, 2012.

embedded systems, on TMote Sky and TMote Invent sensors [10]. We show that ES can be efficiently implemented, using negligible memory and computational requirements. We showcase our implementation with two different demos. The first demo implements a simple environmental monitoring system. A Java interface displays approximated light measurements collected by a wireless node. The second is a labyrinth game, where the player controls a ball on a 3D board by means of wireless inclinometers. We show that up to 90% of communication savings can be achieved using ES and AMS.

We summarize the rationale of the AMS framework in Section 2, and present how exponential smoothing can be implemented in the AMS in Section 3. Section 4 details the environmental monitoring and video gaming implementations.

2 Adaptive Model Selection

Adaptive Model Selection (AMS) [6,7] aims at reducing the radio and energy use of wireless sensors performing periodic data collection tasks, by using predictive models which approximate the real measurements.

Periodic data collection is typical of many wireless sensors' applications. For example, in environmental monitoring applications, the user's interest lies in the evolution of some physical quantity (temperature, humidity, light, ...) at periodic, fixed intervals. In gaming applications such as Wii games, the accelerometer embedded in the Wii controller device sends its measurements to the console at a predefined and high sampling rate. Denoting by $x[t]$ the measurements collected by a sensor at time t, we illustrate periodic data collection in Figure 1. A wireless sensor collects measurements at a predefined sampling rate, and sends them to a *base station*, i.e., a high-end computing unit which will get the same measurements as the sensor. Depending on the applications, the measurements are either displayed to the user (environmental monitoring), or processed for further actions (video gaming).

In many applications, it is often enough to collect an approximation of the sensor measurements. For instance, in plant growth studies, ecologists report that it is sufficient to have an accuracy of ± 0.5°C and 2% for temperature and humidity measurements, respectively [1]. It is therefore not necessary for a sensor to transmit all its measurements. This is the rationale of AMS [7], where *models* are used to approximate the measurements collected by a wireless sensor by means of time series prediction techniques

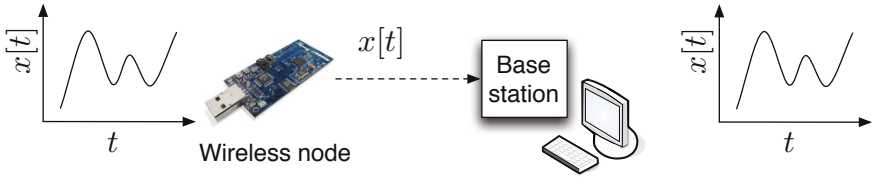

Fig. 1. Periodic data collection: all measurements $x[t]$ are transmitted to the base station, which gets the same measurements as those collected by the wireless sensor

In AMS, a *model* refers to a *parametric function* which predicts, at time $t+m$, $m \in \mathbb{N}^+$, the measurement of the sensor. Formally, the model is a function

$$h_\theta : \mathcal{X} \to \mathbb{R}$$
$$x \mapsto \hat{x}[t+m] = h_\theta(x)$$

where $x \in \mathcal{X}$ is the input to the model (typically a vector of measurements), θ is a vector containing the parameters of the model, and $\hat{x}[t+m]$ is the approximation of the model h to the measurement $x[t+m]$ at time $t+m$.

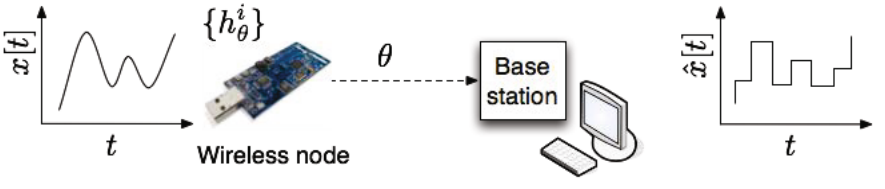

Fig. 2. Approximated data collection: the parameters θ of a predictive model are sent instead of the sensor measurements. Approximations of the measurements are obtained at the base station by means of the predictive model.

In AMS, the parameters of the model are sent instead of the measurements, as illustrated in Fig. 2. The models are estimated by the sensor node on the basis of its past measurements. Given an application-dependent error tolerance ϵ (for example $\epsilon = \pm 0.5°C$), the sensor node can locally assess if the prediction $\hat{x}[t+m]$ made by the model at time $t+m$ is within $\pm\epsilon$ of the true measurement $x[t+m]$. When the prediction is more than ϵ from the real measurement, a new set of model parameters is sent to the base station. At the base station, the model parameters are used to get approximations to the sensor measurements. AMS therefore **guarantees that all the approximated measurements obtained at the base station by means of predictive models are within $\pm\epsilon$ of the real measurements**.

As an example, let us detail the use of AMS with the constant model. The constant model is the most simple predictive model, and was proposed in [8,9]. The model assumes that the measurement at time $t+m$ is the same than that collected at time t, i.e.,

$$h_\theta : \mathcal{X} \to \mathbb{R}$$
$$x \mapsto \hat{x}[t+m] = x[t].$$

With a constant model, there is no model parameter, i.e., θ is empty. Fig. 3 illustrates how a constant model represents a temperature time series. The time series was obtained from the Solbosch Greenhouse of the University of Brussels, during a sunny summer day. 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

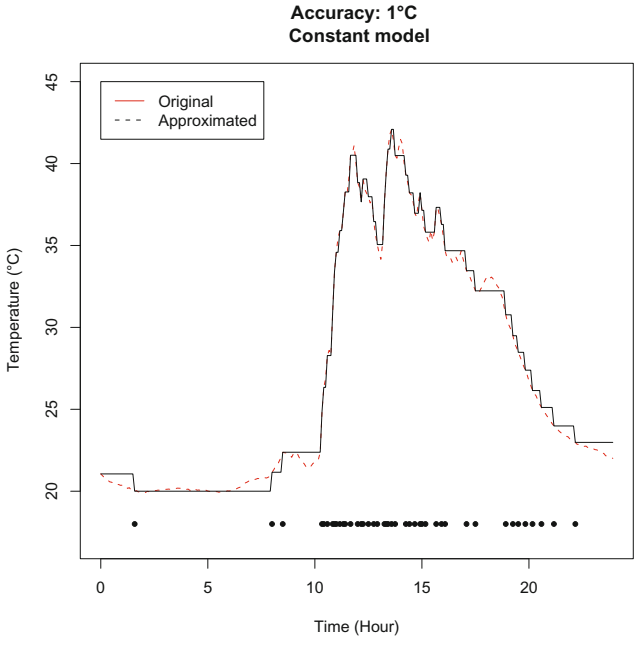

Fig. 3. A constant model acting for a one-day period on a temperature time series from the greenhouse at the University of Brussels, with a constant model and an error threshold set to $\epsilon = 1°C$.

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 AMS, the constant model allows to reduce to 43 the number of measurements transmitted, resulting in about 85% of communication savings.

When the dynamics of the time series is not known *a priori* or in case of non-stationary signals, a set of models with different modeling abilities can be computed and assessed by a sensor node. The set of models is denoted by $\{h_\theta^i\}$, $1 \leq i \leq K$, where K is the number of models computed by the sensor node. The models are all assessed in parallel by the sensor node. When a model update is necessary, the parameters θ of the model that best approximates the sensor's measurements are sent to the base station.

3 Implementation of Exponential Smoothing in AMS

This section presents exponential smoothing (ES) and details its implementation in TinyOS and the adaptive model selection framework.

Exponential Smoothing

Exponential smoothing is a time series prediction technique [3] which has been shown to perform well for a wide variety of time series [2]. Different flavours of

the technique have been proposed. The most simple one is the *simple exponential smoothing*, which consists in a weighted average of the past measurements. The weighted average is computed with

$$s[t] = \alpha x[t-1] + (1-\alpha)s[t-1]$$

where $0 \leq \alpha \leq 1$ is referred to as the *data smoothing factor*. Predictions are given with $\hat{x}[t+m] = s[t]$. Better approximations can usually be obtained using *double exponential smoothing*, with

$$\begin{aligned} s[t] &= \alpha x[t-1] + (1-\alpha)(s[t-1] + b[t-1]) \\ b[t] &= \beta(s[t] - s[t-1]) + (1-\beta)b[t-1] \end{aligned} \tag{1}$$

where $0 \leq \beta \leq 1$ is referred to as the *trend smoothing factor*. Predictions are obtained with

$$\hat{x}[t+m] = s[t] + mb[t]. \tag{2}$$

Note that the simple exponential smoothing is a particular case of double exponential smoothing, with $\beta = 0$. Exponential smoothing is computationally thrifty, which makes it suitable for implementation on resource-constrained wireless sensors.

Implementation in AMS

The use of exponential smoothing requires the specification of the values of the data and trend smoothing factors α and β. For our implementation, we chose to compute models with $\alpha \in \{0.2, 0.4, 0.6, 0.8, 1\}$ and $\beta \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$, so that the wireless node assesses a set of $K = 5 * 6 = 30$ models.

The parameters of the model is the couple $\theta = (s[t], b[t])$ (Eq. 2). Note that θ is sufficient to compute $\hat{x}[t+m]$, $m \in \mathbb{N}^+$, at the base station without knowledge of the true measurements $x[t]$ (See Eq. 2). The initialization is performed at time $t = 1$ by setting $s[1] = x[1]$ and $b[1] = 0$ both on the sensor node and at the base station, for each of the 30 models.

Models are assessed on the sensor node by means of the *relative update rate* [7], which is the average frequency at which packets are sent to the base station. Formally, let $U_i[t]$ be the update rate of model h^i_θ at time t, with $U_i[1] = 1$. When running AMS, the update rate of each model h^i_θ is updated at every time t with

$$U_i[t] = \frac{(t-1) * U_i[t-1] + 1}{t} \tag{3}$$

if model h^i_θ requires to send an update of its parameters θ, and

$$U_i[t] = \frac{(t-1) * U_i[t-1]}{t} \tag{4}$$

otherwise. The relative update rate reflects the percentage of transmitted packets with respect to periodic data collection. Note that the relative update rate for periodic data collection is 1 since it requires the transmission of the totality of

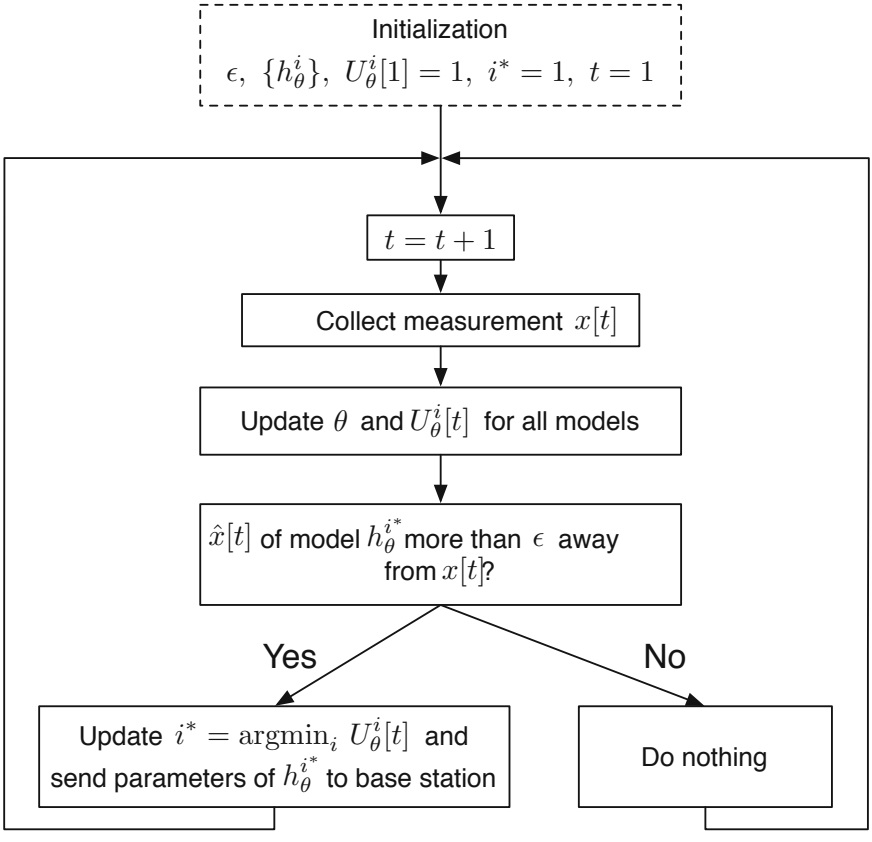

Fig. 4. Adaptive model selection algorithm

the measurements, and that any lower value indicates a gain in the number of transmitted packets. At time t, the best performing model denoted by $h_\theta^{i^*}$ is the one which minimizes the relative update rate, i.e., $i^* = \text{argmin}_i\ U_i[t]$.

At runtime, only one model is shared between the sensor node and the base station. In our implementation, the first shared model is arbitrarily set to $i^* = 1$, i.e., the first model of the collection. Once the approximation for this model is $\pm\epsilon$ away from the real measurement, the node updates i^* to the model which minimizes the relative update rate, and sends the parameters of that model to the base station. A summary of the algorithm is given in Fig. 4.

Energy Efficiency

The rationale of AMS is to compute predictive models in order to reduce radio communication. We motivate in the following that the energy cost incurred by the computation of the predictive models is negligible compared to the energy saved by reducing communication.

The ratio of the energy spent in sending one bit of information to the energy spent in executing one instruction has been estimated to be around 2000 for a variety of real sensor network platforms [11].

Let us first assess the computational overhead of AMS. In the proposed implementation, the sensor node runs and estimates the relative update rate of 30 exponential smoothing models. This requires the sensor node to update, at each time t, the models' parameters (Eq. 1), as well as their relative update rates (Eq. 3 or 4). For one exponential smoothing model, the computational costs of these updates is very low, i.e., on the order of 20 multiplications and additions, depending on the implementation details. For 30 models, the computational overhead of AMS is therefore on the order of 600 CPU cycles.

Let us now assess the communication cost of a packet transmission. We relied in our implementation on TinyOS, a reference operating system for wireless sensor nodes. The standard TinyOS packet structure is detailed in Table 1. The packet overhead, i.e., the extra bytes of information, is 16 bytes. With AMS and exponential smoothing, the packet contains the model parameters $s[t]$ and $b[t]$, each stored on two bytes, giving a total packet size of 20 bytes.

Table 1. Detail of the TinyOS packet structure [14], with the size of each packet component (in bits). The packet overhead is 128 bits, i.e., 16 bytes.

Length	fcfhi	fcflo	dsn	dest	addr	type	group	**Packet content**	str	lqi	crc	ack	time
8	8	8	8	16	16	8	8	**Model size**	8	8	8	8	16

Sending a packet of 20 bytes (160 bits) therefore amounts to $160 * 2000 = 320000$ CPU cycles, i.e. running AMS over about $320000/600 \approx 530$ time instants. This means that if one of the shared predictive model is correct only once every 530 time instants, then AMS provides energy gains. This rough estimate suggests that the energy cost related to running the AMS is in most cases largely compensated by the energy gained in communication savings.

4 Demos and Implementation Details

This section presents two demos in which we apply our implementation of AMS with exponential smoothing. The first demo consists in an environmental monitoring application, and the second a video gaming application. The code for the demos is made available at [16].

Demo 1: Environmental Monitoring

In this demo, a TMote Sky wireless sensor, based on the TelosB prototype platform for research [10], is used to collect light measurements with ES and AMS. The sensor node is programmed in TinyOS v2.1 [14]. A demonstration mode is implemented, allowing to retrieve both the real measurements and the model

parameters on a laptop or desktop computer. The measurements and approximations are displayed by means of a Java interface, see Fig. 5.

The snapshot shows the variations of light measurements (in Lux) in an office exposed to sunlight, for a one minute period, during which the sensor followed a 360 degree rotation around a vertical axis. The sampling rate was of 8Hz (i.e., around 60*8=480 measurement were collected), and the error tolerance $\max_t |\hat{x}[t] - x[t]| = 10$ Lux. AMS identified that, out of the 30 models, $\alpha = 0.2$ and $\beta = 0.6$ provided the most adapted smoothing factors. During that one minute period, almost 90% of the communications could be saved (i.e., around 50 model updates), while providing a qualitatively good approximation of the real measurements as can be seen on the user interface.

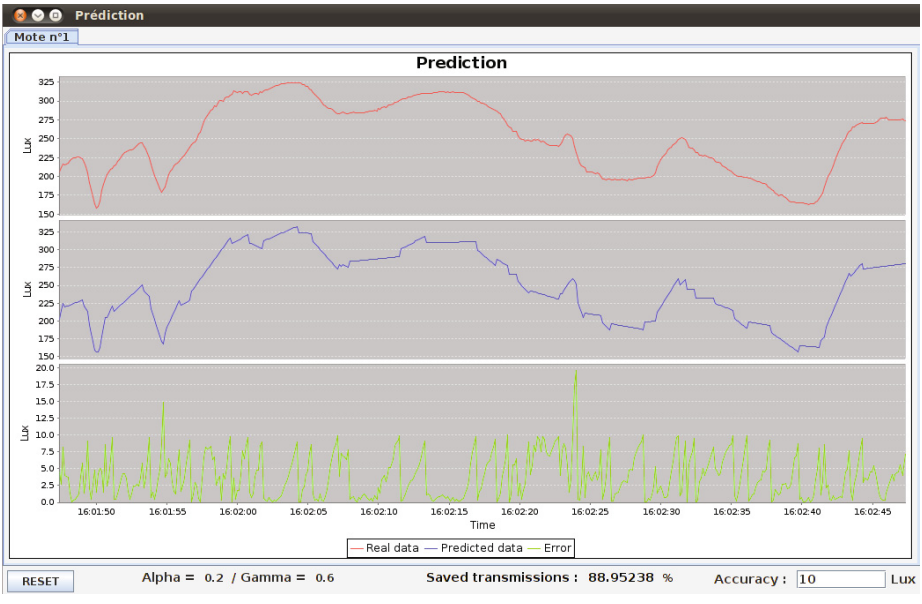

Fig. 5. Environmental monitoring interface displays light measurements collected by a TMote Sky sensor. Top chart shows real measurements, middle chart shows approximated measurements ($\epsilon = 10$ lx), bottom chart shows the predictive model error.

The stripped down version of the TinyOS code for periodic data collection takes 16546 bytes of RAM and 531 bytes of ROM. With AMS, the size of the code increases to 20334 bytes of RAM and 1155 bytes of ROM. AMS was therefore implemented using very little overhead in terms of computational resources.

Demo 2: Video Gaming

In this demo, a TMote Invent sensor node, also based on the TelosB design [10], is used to control a video game with a dual axis wireless inclinometer. The video game is a 3D labyrinth standing on a virtual board (see Fig. 6), where the goal

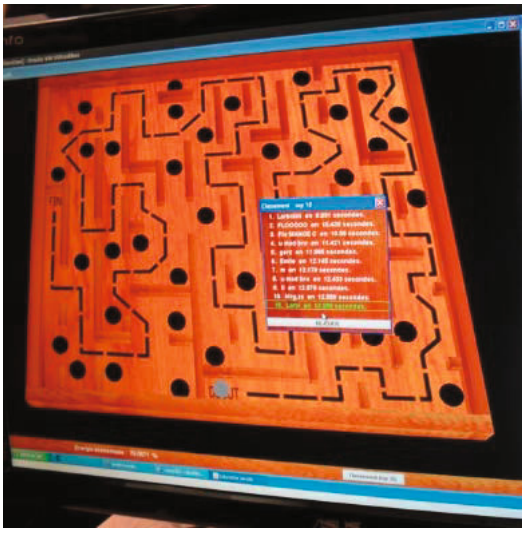

Fig. 6. 3D labyrinth game, controlled by TMote Invent inclinometers

is for the player to move the ball through the labyrinth while avoiding the holes. The game is implemented with Java 3D, and available as open source at [4]. The dual axis inclinometer measurements are used to control the orientation of the board. The node is programmed using Boomerang OS, a TinyOS version specifically designed for TMote nodes [14].

The error tolerance was fixed at 1°, so that approximations did not impair the fluidity and playability of the game. The amount of communications which can be saved depend on how fast the player moves the TMote Invent. Interestingly, since moving the ball through the labyrinth requires to gently control the sensor, we observed that significant communication savings could be saved, reaching more than 90% in most of the games played.

5 Conclusion

In this paper, we discussed an implementation of exponential smoothing in the adaptive model selection framework. The approach allows to perform approximated periodic data collection by relying on time series prediction models. We showed that the method can be implemented using little computational resources, and that significant communication and energy savings could be obtained in practical settings. We illustrated its use on two demos showcasing environmental monitoring and video gaming. In both cases, around 90% communication savings were obtained.

Acknowledgements. This work was supported by the ICT4Rehab project, sponsored by Brussels Institute for Research and Innovation, Innoviris, Belgium. Authors would like to thank the BA3 students of the University of Brussels for their contributions in designing the demos.

References

1. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J., Hong, W.: Model-driven data acquisition in sensor networks. In: VLDB 2004, pp. 588–599 (2004)
2. Gardner Jr., E.S.: Exponential smoothing: The state of the art. Journal of Forecasting 4(1), 1–28 (1985)
3. Holt, C.C.: Forecasting seasonals and trends by exponentially weighted moving averages. International Journal of Forecasting 20(1), 5–10 (2004)
4. Labyrinth 3D game. Java source code. Project Website, `http://www.javafr.com/codes/LABYRINTHE-BILLE-JAVA3D_32818.aspx`
5. Lazaridis, I., Mehrotra, S.: Capturing sensor-generated time series with quality guarantee. In: ICDE 2003, pp. 429–440 (2003)
6. Le Borgne, Y.: Learning in Wireless Sensor Networks for Energy-Efficient Environmental Monitoring. PhD thesis, ULB, Brussels, Belgium (2009)
7. Le Borgne, Y., Santini, S., Bontempi, G.: Adaptive model selection for time series prediction in wireless sensor networks. Journal of Signal Processing 87(12), 3010–3020 (2007)
8. Olston, C., Jiang, J., Widom, J.: Adaptive Filters for Continuous Queries over Distributed Data Streams. In: SIGMOD 2003, pp. 563–574 (2003)
9. Olston, C., Loo, B.T., Widom, J.: Adaptive precision setting for cached approximate values. ACM SIGMOD Record 30, 355–366 (2001)
10. Polastre, J., Szewczyk, R., Culler, D.: Telos: enabling ultra-low power wireless research. In: IPSN 2005, pp. 364–369 (2005)
11. Raghunathan, V.S., Srivastava, C.S.P.: Energy-Aware Wireless Microsensor Networks. IEEE Signal Processing Magazine 19(2), 40–50 (2002)
12. Santini, S.: Adaptive sensor selection algorithms for wireless sensor networks. PhD thesis, ETH Zurich, Zurich, Switzerland (2009)
13. Santini, S., Römer, K.: An adaptive strategy for quality-based data reduction in wireless sensor networks. In: INSS 2006, Chicago, IL, USA, pp. 29–36 (2006)
14. TinyOS. An Open-Source Operating System Designed for Wireless Embedded Sensor Networks. Project Website, `http://www.tinyos.net`
15. Tulone, D., Madden, S.: PAQ: Time Series Forecasting for Approximate Query Answering in Sensor Networks. In: Römer, K., Karl, H., Mattern, F. (eds.) EWSN 2006. LNCS, vol. 3868, pp. 21–37. Springer, Heidelberg (2006)
16. Wireless Lab - University of Brussels. Code and datasets. Project Website, `http://www.ulb.ac.be/di/labo/datasets.html`