Article

Engineering self-adaptive capabilities in wireless sensors based on control algorithms and model-based design methodologies

Raúl Mario del Toro Matamoros [1,*], Andrei Pruteanu [2] and Rodolfo Haber Guerra [1]

[1] Centre for Automation and Robotics, Spanish National Research Council-Technical University of Madrid, 28500 Madrid, Spain; raul.deltoro@car.upm-csic.es (R.M.D.M.); rodolfo.haber@car.upm-csic.es (R.E.H.)
[2] Independent researcher, Romania, andrei.pruteanu@gmail.com
[*] Correspondence: raul.deltoro@car.upm-csic.es

Abstract: The main objective of this work is the design and implementation of self-adaptive capabilities in wireless sensors by applying control engineering and model-based design methodologies. It has been addressed the problem related to the changes in the flow of data packets through the network connection and the excess energy consumption that this causes in these devices. To design the solution, a systemic characterization of the scheduling and execution process of embedded tasks on the device has been carried out. This means defining cause-effect relationships in the system and its modelling theoretically and/or experimentally. In turn, these models facilitate the design of control strategies to improve the dynamic behavior of the system. As a solution, a self-adaptation strategy based on feedforward control algorithm has been designed and developed, which has been applied to improve the dynamic behavior and resource consumption. The developed solution has been satisfactorily evaluated experimentally.

Keywords: self-adaption; wireless sensors; model-based design; control engineering

1. Introduction

Distributed embedded systems constitute an area of growth worldwide, with an increasing number of applications in different domains. Among the applications we can mention, as an example, the monitoring and control systems of large infrastructures, monitoring of logistics systems, monitoring and control applications in high-speed transport systems, public security monitoring systems and monitoring of renewable energy generation systems such as wind or photovoltaic.

However, the introduction of large-scale distributed embedded systems wireless connectivity is hampered by a series of obstacles related to methods and techniques for their design, control and operation. Since the dynamic behavior of these systems is considerably different from the dynamic behavior of centralized systems or distributed systems based on wired devices, the ability to predict the performance of these systems is reduced. This requires that these systems possess special "self-" type capabilities to maintain optimal performance, such as self-adaptation, self-optimization and/or self-organization.

This work is focused on the development of self-adaptive capabilities by applying control engineering and model-based design methodologies. Specifically, it deals with the problem of changes in the flow of data packets through the network connection in wireless devices and the excess energy consumption that this entails. More in specific, the problem has been addressed through the development of a self-adaptive control strategy based on anticipatory control techniques, which has been applied to improve the dynamic behavior of the task scheduling and execution process in distributed embedded systems with low computing capacity.

1.1 Self-adaptation in embedded software

Self-adaptation issues in embedded software have been investigated across different areas of software engineering, such as requirements engineering [1–4], software architecture [5–10], middleware architectures [11–14], component-based development [15–17], model-driven development [18–21] and goal-driven models [22–25]. The majority of these works have been isolated initiatives. In the literature, some techniques for the development of self-adaptive software appear. Some works have obtained interesting results by applying certain techniques, however, other techniques are proposed as initiatives for future work [7].

Despite the innovative nature of the proposed initiatives, they are isolated in nature and do not propose a general methodology for the design of self-adaptation strategies Methodology under which the effects that the introduction of feedback mechanisms may have on the system can also be addressed. Over the years, the software engineering discipline has placed a strong emphasis on the static nature of the architecture of a software system, neglecting to a certain extent the dynamic aspects. In contrast, control engineering places great emphasis on feedback loops [26–28] as a practically essential element to deal with changes in the dynamic behavior of systems and the influence of external disturbances.

However, in the literature there are some works focused on the application of control engineering techniques to introduce self-adaptive capabilities on embedded devices. Just to mention some of them, Cervin et al. [29] have developed a feedback control system with feedforward mechanisms for the scheduling process of tasks dedicated to real-time control duties. The scheduling strategy adjusts the task sampling period based on the estimated measurements of tasks execution time and, also, anticipating changes in tasks workload. The objective of this approach is to improve the performance of the control tasks, obtaining a higher utilization of shared CPU resources. Also, a solution based on a feedback control system has been proposed for multithread applications [30] with the aim to reduce the waste of computational resources for recurrent real-time workloads.

Lindberg et al. [31] have also developed an approach based on feedback control strategies and feedforward actions in order to manage the CPU temperature, in addition to its computational time that is demanded by several tasks with dependencies on multiple resources. The control action has been designed based on a dynamic model of hardware and software resources, which use the flow of resources as a common entity. Moreover, Nayak Seetanadi et al. [32] have designed a solution based on control engineering to optimize both the resource consumption and allocation by the cameras in a video-surveillance system.

The analysis of the literature reveals that the design methodologies applied from the perspective of control engineering, where models-based control strategies and feedforward control actions are used, are viable, and, also, can provide excellent results on its application to the design of self-adaptive mechanisms to embedded software in wireless sensors or devices with low computing capacity.

To design a self-adaptation mechanism by applying a model-based methodology, it has required the following actions:

1. Modelling of the process to be controlled both theoretically and experimentally (see section 2). The objective is to characterize the process applying a systems engineering approach and obtain a useful model for the design of control strategies.
2. Based on the models obtained, design the self-adaptation mechanism based on a feedforward control architecture (see section 3).
3. Implementation and evaluation of the self-adaptive strategy in real devices (see section 4).

2. Theoretical and experimental modelling

This section describes the task scheduling and execution process in embedded systems, both from a theoretical and experimental point of view. The underlying idea is to characterize this process by applying a systems engineering approach and obtain a model that is useful for the design of control strategies.

Characterizing a process as a system means finding its cause-effect relationships. That is, relationships between input and output variables, defined as:

- *Outputs*: measurable or estimable variables related to the requirements or objectives that the system must meet (ex., energy consumption, time consumption, system performance, etc.).
- *Inputs*: independent variables, also measurable, that determine the dynamic behavior of the system (ex., time duration of a basic task execution cycle, listening frequency of the communication channel, number of bytes transmitted and received, CPU cycles consumed by each task, etc.).

2.1. Brief theoretical formalization

In a wireless sensor network, each device constitutes a node that executes a set of tasks that follow a discrete-time execution model. In this execution model, each node executes an approximately constant number of actions or tasks in a given time interval. This time interval is called the execution cycle (EC) or simply cycle. The beginning of each cycle is not synchronized between the nodes of the network (see Figure 1). At the beginning of each cycle, the node sequentially executes tasks such as data acquisition, filtering, merging and communication tasks. At the end of the tasks, the node goes into sleep mode and checks the communication channel at constant time intervals to receive data packets.

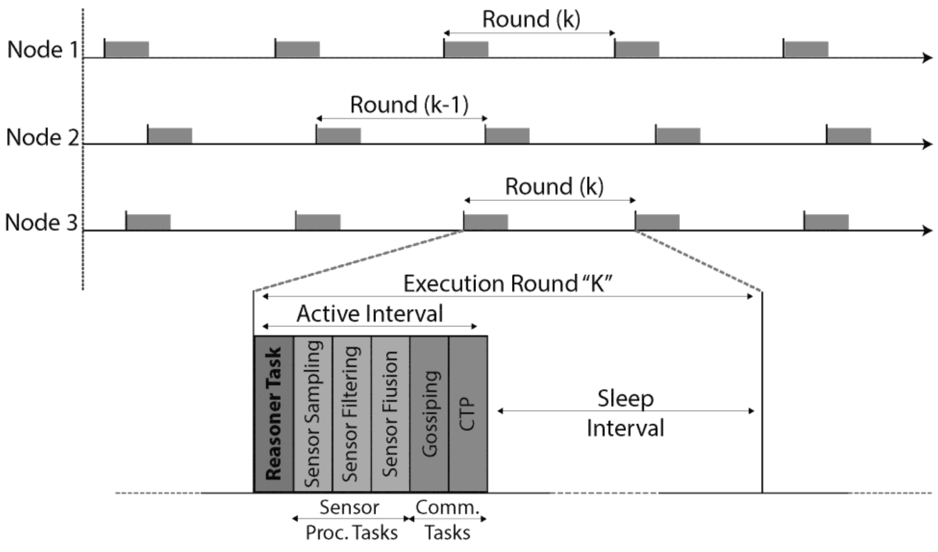


Figure 1. Tasks scheduling and execution process.

In general, the dynamic performance of the task execution process can be characterized through the following variables:

- Time consumed on each execution cycle by each task.
- Energy or power consumed by the hardware components (CPU, transceiver, etc.) when executing those tasks.

The tasks are mainly classified into communication tasks (transmission, reception, etc.) and processing tasks (acquisition, filtering, etc.). A theoretical model of the task execution process is briefly proposed below.

2.1.1 Time consumed by tasks

According to the tasks that are described in the execution model of Figure 1, the time consumed by the tasks, on each execution cycle k, can be expressed in a general way as follows:

- $t_{r_{tx}}$ and $t_{r_{rx}}$, time intervals for receiving and sending packets. These intervals depend mainly on factors such as the communication frequency or bit rate of the transceiver F_b, the number of bytes transmitted (B_{tx}) and received (B_{rx}), communication delays due to the state of the communication channel ($\tau_{r_{tx}}$,$\tau_{r_{rx}}$), communication protocol specifications, among others factors.

$$\begin{gathered} t_{r_{tx}}(k) = f_{tx}\left(B_{tx}, F_b, \tau_{r_{tx}}, k\right) \\ t_{r_{rx}}(k) = f_{rx}\left(B_{rx}, F_b, \tau_{r_{rx}}, k\right) \end{gathered} \tag{1}$$

- $t_{r_{lt}}$, total channel listening time. It is calculated from the period of time (T_{wu}) or frequency (F_{wu}) to listen the channel and the time interval ($\tau_{r_{lt}}$) spend on each period listening the channel.

$$t_{r_{lt}}(k) = f_{r_{lt}}\left(T_{wu}, \tau_{r_{lt}}, k\right) \tag{2}$$

- $t_{p_{tk}}$, computing time used by the processor to execute tasks. This interval depends mainly, among other factors, on the clock frequency of the processor (F_{CPU}) and the number of cycles (C_P) or operations required by the tasks.

$$t_{p_{tk}}(k) = f_{tk}(C_P, F_{CPU}, k) \tag{3}$$

- $t_{s_{acq}}$, time required to acquire samples or measurements. This interval is a function of the number of samples to acquire (S_{acq}), and the time required to acquire a sample ($\tau_{s_{acq}}$).

$$t_{s_{acq}}(k) = f_{acq}\left(S_{acq}, \tau_{s_{acq}}, k\right) \tag{4}$$

2.1.2 Energy and average power consumed on each execution cycle

The total energy E consumed by the device constitutes the integral value of the instantaneous power P consumed by all its hardware components on each execution cycle. In other words, it is mainly the sum of the energy consumed by the transceiver (E_r), the CPU (E_p), the acquisition system (E_s) and other components (E_d):

$$\begin{gathered} E(k) = \int_{t_0(k)}^{t_0(k)+T_{round}} P(t)dt \\ E(k) = E_r(k) + E_p(k) + E_s(k) + E_d(k) \end{gathered} \tag{5}$$

Where the average power consumed in an execution cycle (T_{round}) can generally be expressed as:

$$P(k) = \frac{1}{T_{round}} \int_{t_0(k)}^{t_0(k)+T_{round}} P(t)dt = \frac{E(k)}{T_{round}} \tag{6}$$

Consumed energy also depends on the tasks or operations that each hardware component executes in each execution cycle. For the modeling, the following considerations have been made:

4. The instantaneous current demanded by a certain hardware component during the execution of an operation, will only be considered as appreciable value during the time interval in which the operation or task is executed. The rest of the time that this component is not active, the current consumed will be considered negligible. For example, the instantaneous values of current consumed by the transceiver $I_{r_{tx}}$ during packet transmission are reached only within the time interval $t_{r_{tx}}$ (see equation (1)).
5. In addition, a constant value of power supply voltage V_{cc} to the hardware components is also considered.

Taking into account the above considerations, the energy consumed by each component and the average value of power consumed within the time interval corresponding to the operation, are calculated as follows:

- Consumed energy by the CPU (E_p) and the acquisition system (E_s). They mainly depend from the current consumed when the CPU is activated ($I_{p_{ON}}$) or in standby mode ($I_{p_{sl}}$), from the current consumed by the acquisition system ($I_{s_{acq}}$) and sensors ($I_{s_{ON}}$).

$$E_p(k) = V_{cc}\int_{t_0(k)}^{t_0(k)+T_{round}}\left[I_{p_{ON}}(t) + I_{p_{sl}}(t)\right]dt$$
$$E_p(k) = P_{ON}(k)t_{p_{ON}}(k) + P_{p_{sl}}(k)t_{p_{sl}}(k) \tag{7}$$

$$E_s(k) = V_{cc}\int_{t_0(k)}^{t_0(k)+T_{round}}\left[I_{s_{acq}}(t) + I_{s_{ON}}(t)\right]dt$$
$$E_s(k) = P_{s_{acq}}(k)t_{s_{acq}}(k) + P_{s_{ON}}(k)t_{s_{ON}}(k) \tag{8}$$

- Energy and average power consumed by the transceiver calculated from the current consumed during transmission ($I_{r_{tx}}$), reception ($I_{r_{rx}}$), listening to the communication channel ($I_{r_{sl}}$) or in idle mode ($I_{r_{lt}}$):

$$P_{tx}(k) = \frac{V_{cc}}{t_{r_{tx}}(k)}\int_{t_0(k)}^{t_f(k)} I_{r_{tx}}(t)dt \qquad P_{sl}(k) = \frac{V_{cc}}{t_{r_{sl}}(k)}\int_{t_0(k)}^{t_f(k)} I_{r_{sl}}(t)dt$$
$$P_{rx}(k) = \frac{V_{cc}}{t_{r_{rx}}(k)}\int_{t_0(k)}^{t_f(k)} I_{r_{rx}}(t)dt \quad P_{lt}(k) = \frac{V_{cc}}{t_{r_{lt}}(k)}\int_{t_0(k)}^{t_f(k)} I_{r_{lt}}(t)dt \tag{9}$$

$$E_r(k) = P_{tx}(k)t_{r_{tx}}(k) + P_{rx}(k)t_{r_{rx}}(k) + P_{r_{sl}}(k)t_{r_{sl}}(k) + P_{lt}(k)t_{r_{lt}}(k) \tag{10}$$

where $t_f(k)$ is the finalization time of each round that is calculates as $t_f(k) = t_0(k) + T_{round}$.

2.1. Experimental modelling

The theoretical formalization of the previous section allows us to know which variables and parameters of the task execution process are related to each other. Facilitating the establishment of cause-effect relationships and therefore giving general guidelines of the type of relationship.

But the relationships formulated do not describe specific details such as how a change in the magnitude of an input variable affects the change in magnitude of an output variable. For example, in equation (9) the average power consumed during packets transmission (P_{tx}) depends on the time required by the transceiver to transmit packets ($t_{r_{tx}}$) and on the current ($I_{r_{tx}}$) and voltage consumed during transmission (V_{cc}). However, if this type of information is not known, it can be obtained experimentally.

2.1.2 Experimental setup

In this sense, experiments were carried out to characterize the average power and energy consumption during packets transmission and reception, and, also, by executing acquisition, filtering and merging tasks. The equipment and tools used are detailed below:

- Mobile Device Power Monitor and PowerTool software from Monsoon Solutions Inc.
- 2 SOWNet G-Node G301 Wireless Sensor Node: one of the sensors acts as a packet sender and the other as a receiver.
- Arduino board to mark specific events through the GPIO pins.
- Matlab as a tool for data analysis, processing and modelling.

(**a**) (**b**)

Figure 2. Experimental setup devices: (**a**) Monsoon Mobile Device Power Monitor; (**b**) SOWNet G-Node G301 Wireless Sensor.

The experimental parameters used are listed below. For each of the listening frequency values of the receiving node, packets of different lengths were sent:

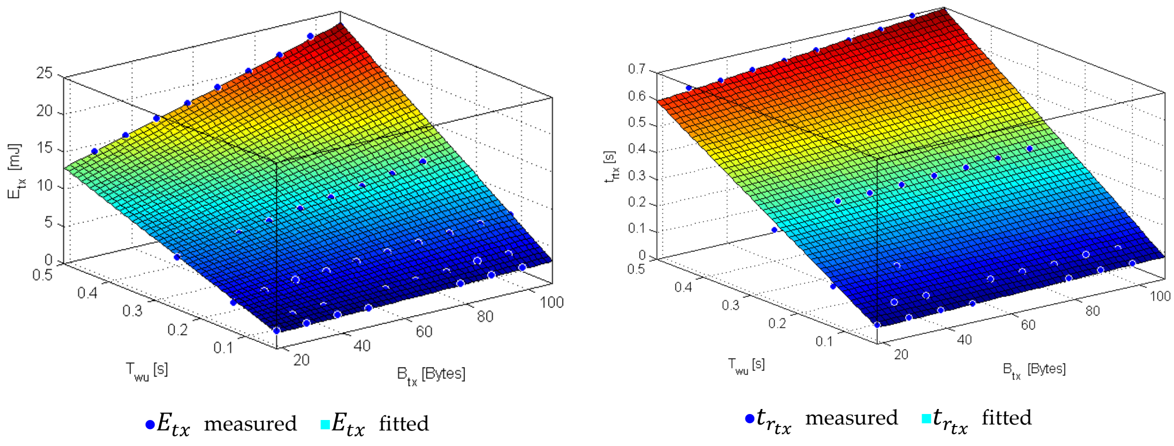
- **Number of transmitted Bytes or packet Size**: 18, 28, 38, 48, 58, 68, 78, 88, 98, 108.
- **Listening frequency (Hz)**: 2, 4, 8, 16, 32.

2.1.3 Model identification

The data obtained experimentally were analyzed and processed using Matlab. Applying regression techniques, for each of the types of events, models of energy consumed and time per execution cycle were obtained.

In the models, n_{tx} and n_{rx} are defined as the total number of sequences for transmission and reception events, respectively, that occur on each execution cycle k. The models obtained, their parameters obtained for a 95% confidence bounds and coefficient of determination R^2, are summarized below.

- **Packets transmission**



(**a**) (**b**)

Figure 3. Fitted models for packets transmission: (**a**) Energy during transmission; (**b**) Time during transmission.

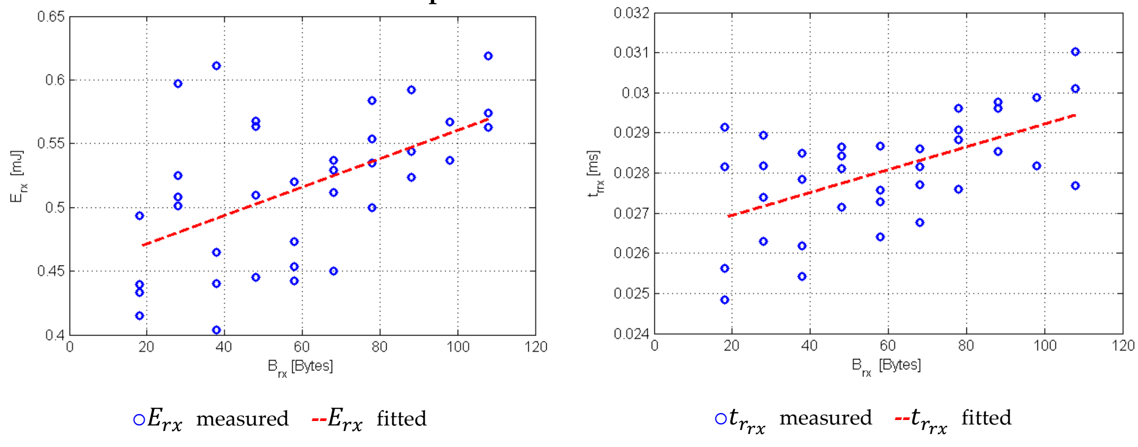
$$t_{rtx}(k) = g_{tx}(B_{tx}, n_{tx}, T_{wu}, k) \\ = (\tau_{atx} + \tau_{btx}T_{wu})B_{tx}(k) + n_{tx}(\tau_{ctx}T_{wu} + \tau_{dtx}) \tag{11}$$

where $\boldsymbol{\tau_{atx}}$ = 0.0001722, τ_{btx} = 0.001945, τ_{ctx} = 1.098, τ_{dtx} = 0.02437 ($R^2 = 0.9992$).

$$E_{tx}(k) = f_{tx}(B_{tx}, n_{tx}, T_{wu}, k) \\ = (\varepsilon_{atx} + \varepsilon_{btx}T_{wu})B_{tx}(k) + n_{tx}(\varepsilon_{ctx}T_{wu} + \varepsilon_{dtx}) \tag{12}$$

where ε_{atx} = 0.002544, ε_{btx} = 0.2316, ε_{ctx} = 18.68, ε_{dtx} = 1.399 (R^2 = 0.9992).

- **Packets reception**



(**a**) (**b**)

Figure 4. Fitted models for packets reception: (**a**) Energy; (**b**) Time.

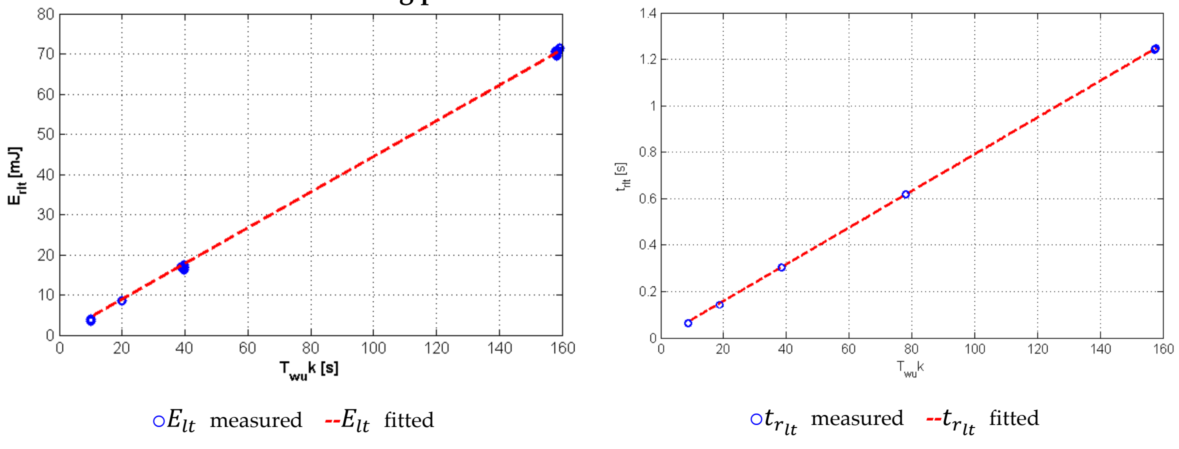
$$t_{r_{rx}}(k) = g_{rx}(B_{rx}, n_{rx}, k) = \tau_{arx}B_{rx}(k) + n_{rx}\tau_{brx} \quad (13)$$

where $\boldsymbol{\tau_{arx}}$ = 2.876e-005, τ_{brx} = 0.02634 (R^2 = 0. 3364).

$$E_{rx}(k) = f_{rx}(B_{rx}, n_{rx}, k) = \varepsilon_{arx}B_{rx}(k) + n_{rx}\varepsilon_{brx} \quad (14)$$

where ε_{arx} = 0.001091, ε_{brx} = 0.4509 (R^2 = 0. 2750).

- **Listening period**



(a) (b)

Figure 5. Fitted models for the listening period: (**a**) Energy; (**b**) Time.

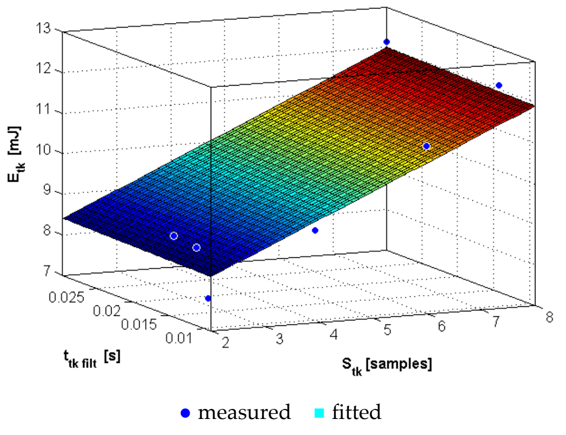
$$t_{r_{lt}}(k) = g_{r_{lt}}\left(T_{wu}, t_{r_{tx}}, t_{r_{rx}}, k\right) = \tau_{r_{lt}}\frac{\left(T_{round} - t_{r_{tx}}(k) - t_{r_{rx}}(k)\right)}{T_{wu}} \quad (15)$$

where $\tau_{r_{lt}}$ = 0.01061 (R^2 = 0. 9997).

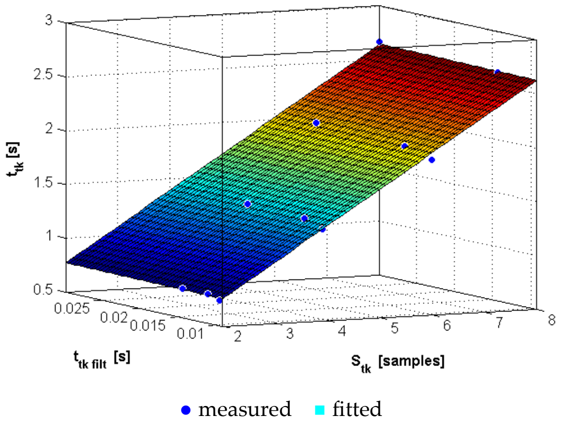
$$E_{lt}(k) = f_{lt}(T_{wu}, k) = \varepsilon_{alt}\frac{\left(T_{round} - t_{r_{tx}}(k) - t_{r_{rx}}(k)\right)}{T_{wu}} \quad (16)$$

where ε_{alt} = 0.4435 (R^2 = 0. 9996).

- **Acquisition and data filtering tasks**



(**a**)



(**b**)

Figure 6. Fitted models for acquisition and data filtering tasks: (**a**) Energy; (**b**) Time.

$$\begin{aligned} t_{p_{tk}+s_{acq}}(k) &= g_{tk+acq}\left(S_{acq},\tau_{s_{acq}},C_P,F_{CPU},k\right) \\ &= \tau_{s_{acq}}S_{acq}(k) + F_{CPU}{}^{-1}C_{P_{tkf}}(k) + \tau_{tk} \end{aligned} \tag{17}$$

where $\tau_{s_{acq}}$ = *0.3123*, τ_{tk} = *0.1199 (R² = 0. 9981).*

$$\begin{aligned} E_{tk+acq}(k) &= f_{tk+acq}\left(S_{acq},C_{P_{tk}},k\right) \\ &= \varepsilon_{atk}S_{acq}(k) + \varepsilon_{btk}F_{CPU}{}^{-1}C_{P_{tk}}(k) + \varepsilon_{ctk} \end{aligned} \tag{18}$$

where ε_{atk} = *0.5987*, ε_{btk} = *3.797*, ε_{ctk} = *7.093 (R² = 0. 9152).*

The total energy consumed in each execution cycle is variable because it depends on the tasks or events that are executed in each cycle. However, the tasks are executed periodically every a certain number of execution cycles. The execution period for these tasks can be expressed in the form $l = L_k k$, being L_k the periodicity relation with respect to the basic execution cycle k. Therefore, the energy consumed $E(l)$, its temporary duration $T_{l\_rnd}(l)$ and average power consumed $P(l)$ for the execution period l can be expressed as follows:

$$\begin{gathered} T_{l\_rnd}(l) = \sum_{k=k_0}^{k=k_0+L_k} T_{round}(k) \\ E(l) = \sum_{k=k_0}^{k=k_0+L_k} E(k)\,; P\,(l) = \frac{E(l)}{T_{l\_rnd}(l)} \end{gathered} \tag{19}$$

3. Design of a control engineering-based self-adaptive strategy

It is proposed to design a self-adaptation strategy for the process of planning and executing tasks based on the feedforward control architecture (see Figure 7). Feedforward control is a model-based method that provides an open-loop control compensation action (feedforward action). The design of this control action is much simpler than a more complex closed-loop control action. In addition, it would avoid the possible effects that could be introduced into the dynamics of the system as a result of the negative feedback.

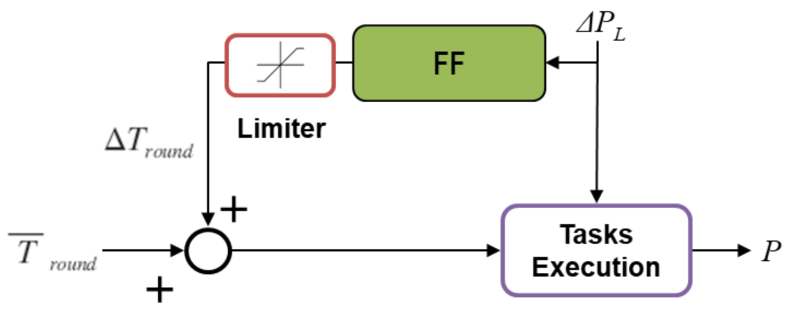


Figure 7. Self-adaptive strategy based on the feedforward control architecture.

In general, the design of the control strategy is based on the following premises:

- It would aim to cancel or minimize the effects of disturbances or load on the average power consumption through slight corrections ($\varDelta T_{round}$) of the execution cycle time period (control action).
- The load on the average power consumption ($\varDelta P_L$) of the sensor is calculated as the difference over its nominal or expected value during a certain period of time l.
- The controller will only compensate loads greater than zero, that is, consumption above the nominal value, but in a limited way. The objective would be not to excessively modify the nominal value of the duration time of a task execution cycle ($\bar{T}_{round}$).
- The energy consumption will be estimated from the obtained experimental models presented in previous sections. Specifically the models that estimate consumption during the reception and transmission of packets, execution of data acquisition tasks and filtering.

The steps performed to obtain the equation of the control strategy are described below:

1. Define the equation of average power consumed $P(l)$ during the period of time l, considering the nominal value for that period $\bar{P}_l$ and the load on the average power consumption $\varDelta P_L$.

$$P(l) = \bar{P}_l + \varDelta P_L \tag{20}$$

2. Calculate the nominal average power consumed $\bar{P}_l$ during the cycle l (see equation (19)) based on the nominal values of the estimated energy consumed $\bar{E}_l$, the cycle duration time $\bar{T}_{l\_rnd}$ and the nominal values of the estimated energy consumed on a simple execution cycle $\bar{E}_k$ and its duration time $\bar{T}_{round}$.

$$\bar{P}_l = \frac{\bar{E}_l}{\bar{T}_{l\_rnd}};\ \bar{E}_l = \sum_{k=k_0}^{k=k_0+L_k} \bar{E}_k\ ;\ \bar{T}_{l\_rnd} = L_k \bar{T}_{round} \tag{21}$$

3. Define the equations of the feedforward control action.

$$\begin{gathered} \varDelta P_L = P(l) - \bar{P}_l = 0 \\ E(l)\bar{T}_{l\_rnd} - \bar{E}_l T_{l\_rnd}(l) = 0 \end{gathered} \tag{22}$$

4. Introduce the control variable $\varDelta T_{round}(k)$ and define the equation for the calculation of the control action for a simple execution cycle.

$$\begin{gathered} T_{l\_rnd}(l) = L_k(\bar{T}_{round} + \varDelta T_{round}(k)) \\ \varDelta T_{round}(k) = \frac{(E(l) - \bar{E}_l)}{\alpha_{ff}};\ \alpha_{ff} = \frac{\bar{E}_l}{\bar{T}_{round}} \end{gathered} \tag{23}$$

5. Considering the introduction of a limiter for the magnitude of the control action, then final equation for the control action can be expressed as follows:

$$\Delta T_{round}(k) = \begin{cases} |\Delta T_{round}^{LIM}|; & \Delta T_{round}(k) \geq |\Delta T_{round}^{LIM}| \\ \dfrac{(E(l) - \bar{E}_l)}{\alpha_{ff}}; & -|\Delta T_{round}^{LIM}| < \Delta T_{round}(k) < |\Delta T_{round}^{LIM}| \\ -|\Delta T_{round}^{LIM}|; & \Delta T_{round}(k) \leq -|\Delta T_{round}^{LIM}| \end{cases} \tag{24}$$
$$|\Delta T_{round}^{LIM}| = \Delta T_{LIM}\bar{T}_{round}\,; 0 < \Delta T_{LIM} < 1$$

4. Implementation and evaluation of the self-adaptive strategy

The evaluation tests were carried out according to the following experimental setup and design:

- 6 wireless sensors were deployed in order to measure temperature inside a room: sensor models SOWNET GNODE with Contiki-OS-2.7 operating system embedded.
- The control strategy has been implemented in C language and, mainly, two functions have been implemented
  1) *feedForward()*: function to calculate the feedforward control action.
  2) *getEnergyConsumption()*: function to calculate the estimated consumed energy.
- Two of the nodes were powered with supercapacitors in such a way that the changes that the control system could introduce in the energy consumption of the sensors could be evaluated in a short period of time.
  i. The control algorithm was activated in the node called NC_ON, so its execution cycle period is variable.
  ii. In another node named NC_OFF the control algorithm was disabled and the task execution cycle period was set to 5 seconds.
- The rest of the nodes use a randomly variable execution cycle time in order to introduce disturbances in network communications and therefore overload the number of packets expected by the control algorithm. The execution cycle time varies randomly in a range between 4.5 and 5.5 seconds (±10% around nominal value).
  - Period greater than 5 seconds implies fewer packets sent than expected.
  - A period less than 5 seconds implies an increase in the number of packets sent over what was expected.
- For each execution cycle, the following information stored in the nodes is acquired through an FTDI cable: duration of the execution cycle, supply voltage, estimated energy consumption, number of packets sent and received, among other variables.
- Use of the variation rate (or battery discharge rate) of the power supply voltage of the node per unit of time, as a reference variable to evaluate the variation in the average power consumption of the node. This rate is proportional to the current supplied by the battery or super-capacitor, which in turn corresponds to the variation on its charge.
- A total of three experimental tests were performed, for which two different channel listening frequencies were used (experiment 1 frequency of 28 Hz and 2 Hz in the other 2 experiments).

Table 1 summarizes the data of the experiments carried out where:

- Third and fourth columns represent the mean value of the battery discharge rate for both nodes (NC_ON and NC_OFF), which in turn corresponds to the average slope of the supply voltage of the nodes.
- Fourth column also represents the discharge rate but for the node where the control algorithm was disabled and a fixed task execution cycle time period (NC_OFF).
- The last column is the percentage difference in the discharge rate between both nodes which is calculated as follows

$$\delta = 100\left|\frac{dV/dt_{NC\_OFF} - dV/dt_{NC\_ON}}{dV/dt_{NC\_OFF}}\right| \tag{25}$$

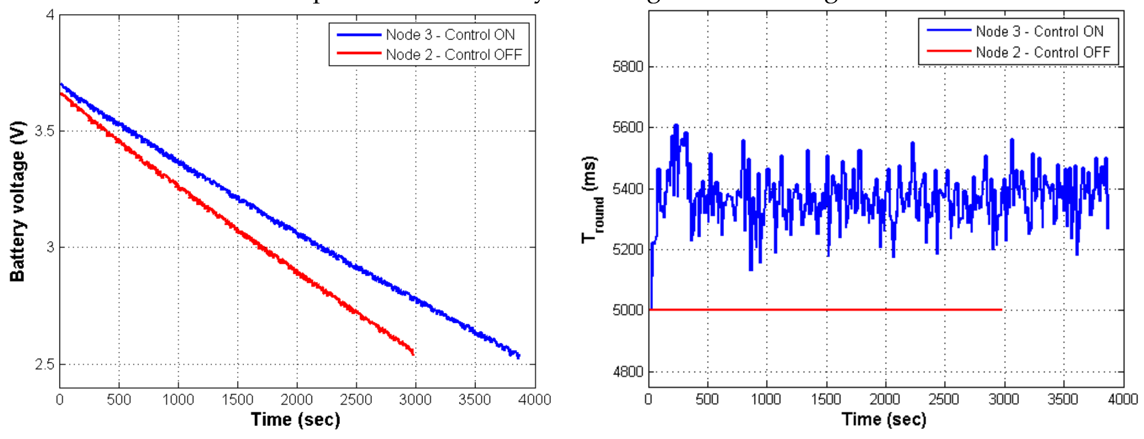
- The last row is the mean value of the discharge rate difference of the three experiments.

Table 1. Summary of experimental data.

Exp. Nº	Channel listening frequency [Hz]	NC_ON dV/dt [mV/s]	NC_OFF dV/dt [mV/s]	Discharge rate difference δ [%]
1	28	-0.31893	-0.39127	18.49
2	2	-0.30852	-0.38576	20.02
3	2	-0.30147	-0.37877	20.41
			Mean value	**19.64**

Figure 8 depicts the behavior of the supply voltage and the duration of the execution cycle for the nodes NC_ON and NC_OFF during experiment number 3. Because the feedforward control algorithm is active in the NC_ON node, the execution cycle varies to compensate the overload of packets that are received and thus the average power consumption. The slope of the supply voltage curve is less for the NC_ON node compared to the NC_OFF node. This means that the discharge rate of the supply voltage for the NC_ON node is lower and therefore the average power consumption is also lower. The average values of the discharge rate for both nodes, as well as the percentage difference between the two nodes, are detailed in Table 1.

The average value of the difference between both nodes for all the experiments is also specified, which has a value of 19.64% and, in turn, represents an increase of the same value in the battery charge. These data demonstrate the improvements in power consumption introduced by the designed control algorithm.



(a) (b)

Figure 8. Data from the evaluation tests corresponding to Exp. Nº 3: **a)** Supply voltage; **b)** execution cycle duration.

5. Conclusions and future works

In this work, self-adaptive capabilities in wireless sensors have been developed by applying control engineering and model-based design methodologies, as is the case of the feedforward control strategy. The proposed design methodology would enable designers to apply analytical methods, more appropriate in control engineering, to develop self-adaptive strategies for the process of tasks scheduling and execution for wireless sensor networks. The approach used differs from ad hoc approaches, typically used by software engineers, which rely on numerous iterations during the design stage. Compared with these approaches, the proposed approach based on control engineering techniques can reduce the design time of self-adaptive capabilities in wireless sensor networks.

However, despite the results obtained, some open questions would remain that are listed below:

- From the dynamic point of view, how does the proposed solution affect the rest of the nodes in the network?
- Instead of using the execution cycle time as control action, what would be the result obtained if other variables were used, for example the channel listening frequency?
- Would other solutions based on distributed decision making, for example as is the case of algorithms based on consensus, be able to improve the performance of the system in a general way?
- Would it be useful to introduce the results achieved (models, algorithms, etc.) in design tools used by designers of wireless sensor networks with self-adaptive capabilities?

These and other possible questions would be interesting topics to study and develop in future works. The results achieved in this work can serve as a basis for developing tools to support designers of wireless sensor networks with self-adaptive capabilities. This type of tools can be transferred to the industry related to Information and Communications Technologies, so it would have a high economic and social impact. This is mainly due to the importance that Cyber-Physical Systems and the Internet of Things are gaining more and more in different industrial sectors.

Author Contributions: Conceptualization, methodology, investigation, validation, Raúl Mario del Toro Matamoros and Andrei Pruteanu; software and article review, Andrei Pruteanu; formal analysis, writing—original draft preparation, writing—review and editing, Raúl Mario del Toro Matamoros; supervision, project administration, funding acquisition and article review, Rodolfo Haber Guerra. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the project "Self-reconfiguration for Industrial Cyber-Physical Systems based on digital twins and Artificial Intelligence. Methods and application in Industry 4.0 pilot line (SELFRECO)", grant ID PID2021-127763OB-100, supported by Ministerio de Ciencia e Innovación, Spain

Data Availability Statement: Not applicable.

Acknowledgments: Authors would to acknowledge the support given by researchers from the Embedded and Networked Systems, at Technical University of Delft during the experimental phase.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Samin, H.; Bencomo, N.; Sawyer, P. Decision-Making under Uncertainty: Be Aware of Your Priorities. *Softw Syst Model* **2022**, doi:10.1007/s10270-021-00956-0.
2. Wohlrab, R.; Garlan, D. A Negotiation Support System for Defining Utility Functions for Multi-Stakeholder Self-Adaptive Systems. *Requir Eng* **2022**, doi:10.1007/s00766-021-00368-y.
3. Sawyer, P.; Bencomo, N.; Whittle, J.; Letier, E.; Finkelstein, A. Requirements-Aware Systems: A Research Agenda for RE for Self-Adaptive Systems. In Proceedings of the 2010 18th IEEE International Requirements Engineering Conference; IEEE, September 2010; pp. 95–103.
4. Brown, G.; Cheng, B.H.C.; Goldsby, H.; Zhang, J. Goal-Oriented Specification of Adaptation Requirements Engineering in Adaptive Systems. In Proceedings of the Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems; 2006; pp. 23–29.
5. den Hamer, P.; Skramstad, T. Autonomic Service-Oriented Architecture for Resilient Complex Systems. In Proceedings of the Proceedings of the IEEE Symposium on Reliable Distributed Systems; 2011; pp. 62–66.

6. Oreizy, P.; Gorlick, M.M.; Taylor, R.N.; Heimbigner, D.; Johnson, G.; Medvidovc, N.; Quilici, A.; Rosenblum, D.S.; Wolf, A.L. An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems and Their Applications* **1999**, *14*, 54–62, doi:10.1109/5254.769885.
7. Macías-Escrivá, F.D.; Haber, R.; del Toro, R.; Hernandez, V. Self-Adaptive Systems: A Survey of Current Approaches, Research Challenges and Applications. *Expert Syst Appl* **2013**, *40*, 7267–7279, doi:10.1016/j.eswa.2013.07.033.
8. Richter, U.; Mnif, M.; Brank, J.; Müller-Schloer, C.; Schmeck, H. Towards a Generic Observer/Controller Architecture for Organic Computing. In Proceedings of the INFORMATIK 2006 - Informatik fur Menschen, Beitrage der 36. Jahrestagung der Gesellschaft fur Informatik e.V. (GI); 2006; Vol. 1, pp. 112–119.
9. Garlan, D.; Cheng, S.-W.; Schmerl, B. *Increasing System Dependability through Architecture-Based Self-Repair*; 2003; Vol. 2677 LNCS; ISBN 9783540407270.
10. Dobaj, J.; Riel, A.; Krug, T.; Seidl, M.; MacHer, G.; Egretzberger, M. Towards Digital Twin-Enabled DevOps for CPS Providing Architecture-Based Service Adaptation &amp; Verification at Runtime. In Proceedings of the Proceedings - 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2022; 2022; pp. 132–143.
11. Schmitt, J.; Roth, M.; Kiefhaber, R.; Kluge, F.; Ungerer, T. Realizing Self-x Properties by an Automated Planner. In Proceedings of the Proceedings of the 8th ACM International Conference on Autonomic Computing, ICAC 2011 and Co-located Workshops; 2011; pp. 185–186.
12. Geihs, K.; Barone, P.; Eliassen, F.; Floch, J.; Fricke, R.; Gjorven, E.; Hallsteinsen, S.; Horn, G.; Khan, M.U.; Mamelli, A.; et al. A Comprehensive Solution for Application-Level Adaptation. *Softw Pract Exp* **2009**, *39*, 385–422, doi:10.1002/spe.900.
13. Liu, H.; Parashar, M. Accord: A Programming Framework for Autonomic Applications. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* **2006**, *36*, 341–352, doi:10.1142/S0219493706001761.
14. Niemczyk, S.; Opfer, S.; Fredivianus, N.; Geihs, K. ICE-Self-Configuration of Information Processing in Heterogeneous Agent Teams. In Proceedings of the Proceedings of the ACM Symposium on Applied Computing; 2017; Vol. Part F1280, pp. 417–423.
15. Bencomo, N.; Grace, P.; Flores, C.; Hughes, D.; Blair, G. Genie: Supporting the Model Driven Development of Reflective, Component-Based Adaptive Systems. In Proceedings of the Proceedings - International Conference on Software Engineering; 2008; pp. 811–814.
16. Peper, C.; Schneider, D. Component Engineering for Adaptive Ad-Hoc Systems. In Proceedings of the Proceedings - International Conference on Software Engineering; 2008; pp. 49–56.
17. Huynh, N.-T. State Transfer Management in Adaptive Software: An Approach from Design to Runtime. In Proceedings of the RIVF 2019 - Proceedings: 2019 IEEE-RIVF International Conference on Computing and Communication Technologies; 2019.
18. Vogel, T.; Neumann, S.; Hildebrandt, S.; Giese, H.; Becker, B. Model-Driven Architectural Monitoring and Adaptation for Autonomic Systems. In Proceedings of the Proceedings of the 6th International Conference on Autonomic Computing, ICAC’09; 2009; pp. 67–68.
19. Vogel, T. MRUBiS: An Exemplar for Model-Based Architectural Self-Healing and Self-Optimization. In Proceedings of the Proceedings - International Conference on Software Engineering; 2018; pp. 101–107.
20. Pol, M.; Diaconescu, A. Multi - Level Online Learning and Reasoning for Self-Integrating Systems. In Proceedings of the Proceedings - 2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion, ACSOS-C 2021; 2021; pp. 187–192.

21. van Leeuwen, C.J.; de Gier, J.M.; de Filho, J.A.O.D.; Papp, Z. Model-Based Architecture Optimization for Self-Adaptive Networked Signal Processing Systems. In Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems, SASO; 2014; Vol. 2014-Decem, pp. 187–188.
22. Salehie, M.; Tahvildari, L. Towards a Goal-Driven Approach to Action Selection in Self-Adaptive Software. *Softw Pract Exp* **2012**, *42*, 211–233, doi:10.1002/spe.1066.
23. Emami-Taba, M. Decision-Making in Self-Protecting Software Systems: A Game-Theoretic Approach. In Proceedings of the Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017; 2017; pp. 77–79.
24. Bowers, K.M.; Fredericks, E.M.; Cheng, B.H.C. *Automated Optimization of Weighted Non-Functional Objectives in Self-Adaptive Systems*; 2018; Vol. 11036 LNCS; ISBN 9783319992402.
25. Thomas, C.; Mirzaei, E.; Wudka, B.; Siefke, L.; Sommer, V. *Service-Oriented Reconfiguration in Systems of Systems Assured by Dynamic Modular Safety Cases*; 2021; Vol. 1462; ISBN 9783030865061.
26. Tanner, J.A. Feedback Control in Living Prototypes: A New Vista in Control Engineering. *Medical Electronics &amp; Biological Engineering* **1963**, *1*, 333–351, doi:10.1007/BF02474417.
27. Hellerstein, J.L.; Diao, Y.; Parekh, S.; Tilbury, D.M. *Feedback Control of Computing Systems*; John Wiley & Sons, 2004; ISBN 0471668818.
28. Franklin, G.F.; Powell, J.D.; Emami-Naeini, A. Feedback Control of Dynamic Systems. **2019**.
29. Cervin, A.; Eker, J.; Bernhardsson, B.; Årzén, K.-E. Feedback-Feedforward Scheduling of Control Tasks. *Real-Time Systems* **2002**, *23*, 25–53, doi:10.1023/A:1015394302429.
30. Papadopoulos, A.V.; Agrawal, K.; Bini, E.; Baruah, S. Feedback-Based Resource Management for Multi-Threaded Applications. *Real-Time Systems* **2022**, doi:10.1007/s11241-022-09386-7.
31. Lindberg, M.; Årzén, K.-E. Feedback Control of Cyber-Physical Systems with Multi Resource Dependencies and Model Uncertainties. In Proceedings of the Proceedings - Real-Time Systems Symposium; 2010; pp. 85–94.
32. Nayak Seetanadi, G.; Årzen, K.-E.; Maggio, M. Control-Based Event-Driven Bandwidth Allocation Scheme for Video-Surveillance Systems. *Cyber-Physical Systems* **2022**, *8*, 111–137, doi:10.1080/23335777.2021.1887364.