Chapter 1
Introduction

The emergence of nontrivial embedded sensor units (e.g., those embedded in smartphones and other everyday appliances), networked embedded systems, and sensor/actuator networks (e.g., those associated with services running on mobiles and wireless sensor networks) has made possible the design and implementation of several sophisticated applications where large amounts of real-time data are collected to constitute a *big data* picture as time passes. Acquired data are then processed at local, cluster-of-units or server level to take the appropriate actions or make the most appropriate decision.

Acquired data may be influenced by uncontrolled variables as well as external environmental factors, which impair the availability, validity, and usability of data. Sensors are affected by uncertainty and faults, either transient or permanent, negatively impacting on the subsequent decision-making process. Moreover, finite precision representation, algorithm pruning at the numerical computational level so as to satisfy execution time and memory constraints, and the use of algorithms and parameters learned from data, introduce additional levels of uncertainty affecting the accuracy of the decision-making algorithm.

On the technological side, advances in embedded processor have made available embedded systems with a computational capacity ready to support sophisticated intelligent mechanisms and the ability to host a plethora of sensors, hence providing a new degree of freedom to applications. For instance, a smartphone might host an ambient light sensor for adjusting the display brightness (which in turn saves battery power), a proximity sensor (e.g., to switch off the display), a MEMS accelerometer (e.g., to commute a picture visualization from landscape to portrait), a compass (to identify the magnetic pole and orientate maps to the north), and GPS (to provide the local coordinates). It should be emphasized that sensors can even be virtual, e.g., a sensor that detects a change in inclination by processing the output of an existing tri-axial MEMS accelerometer, a sensor providing the profiling of a web user, or one proposing email features for subsequent spam detection.

Interestingly, the adjective *intelligent*, when associated with a sensing system, can be inflected differently, depending on the reference community. As such, it may in some way imply: the ability to make decisions, the capability of learning from external stimuli, the promptness in adapting to changes, or the possibility of executing computationally intelligent algorithms. All the above definitions, explicitly or implicitly, rely on a computational paradigm or application which receives and processes incoming acquisitions to accomplish the requested task. Under this framework, the literature generally assumes that sensors are fault free, that data are stationary, time invariant, available, and ready to be used, and that the application is capable of providing outputs and taking decisions. Unfortunately, assumptions about the quality and validity of data are so implicitly taken as valid by scholars that, most of the time, even their existence as assumptions is forgotten.

From the perspective of the designer and the user, we expect that the coded application is well performing and fully satisfies technological and application constraints, e.g., power consumption, cost, execution time, and accuracy performance. But how do we assess the quality of the algorithm given the different degrees of freedom available in the design space of an unknown solution to a given problem? We recall that numerical embedded applications operate in an uncertainty-affected world: do we really need to waste resources to represent and process uncertainty? We should balance complexity with accuracy.

Moreover, we are interested in designing an application characterized by robust features, so that perturbations affecting its computation and the presence of uncertainty will be tolerated well and their effects mitigated. How can we evaluate an effective index for the application robustness allowing us to select/drive the design phase toward the most appropriate solution? Owing to the different forms of uncertainty affecting the numerical computation, the code execution provides an output that is approximated. If we investigate this concept a bit further, we discover that the natural way of carrying out a numerical embedded computation is to provide a result that approximates the correct one with high probability. That is what the Probably Approximately Correct Computation (PACC) is about.

Adaptation is another main feature any intelligent system should possess. It represents the lowest form of intelligence associated with passive, uncontrolled intelligent mechanisms, like those of emotional processing in the human brain. Some of the adaptation mechanisms are well known and operate at the hardware level to keep control of the power consumption, e.g., by means of voltage and clock frequency scaling. However, adaptation must be intended in a broader sense within an embedded framework, with learning mechanisms representing the main tools to keep track of the environmental evolution. Although adaptation plays a relevant role in managing evolving applications, it may not be sufficient in those applications requiring more sophisticated responses to meet a higher Quality of Service (QoS) and performance. Here, cognitive mechanisms must be envisaged, which act as controlled conscious processes. The basics of cognition, as well as some implications on existing mechanisms specified in the book, will be addressed since it is strongly perceived that the next generation of embedded systems will be based on cognitive-based approaches.

Another issue is that of learning in a nonstationary environment. For most applications, we assume that the data stream to be processed is time invariant and, hence, a time invariant-based application is designed to solve the problem. Although the time invariant hypothesis might hold for a short period of time, it probably does not in the mid term and never in the long run. We should not be surprised, our body evolves over time, so embedded systems and the way they interact thanks to sensors and actuators with the external environment do. This is a consequence of aging effects at the sensor level, faults or changes affecting the environment, and/or the interaction between the embedded system and the environment. Such mechanisms induce changes in the structure of the process generating the data, that evolves, with the consequence that the application being executed rapidly becomes obsolete unless learning strategies are taken into account. Learning in an evolving environment aims at addressing this aspect so that the information carried by incoming data is not only used for decision making but, also, to keep track of the changes and react accordingly. When the change is of a fault-type, we need to intervene with suitable fault diagnosis procedures whose effectiveness depends on a priori information about the environment, the nature of the process generating measurements to be acquired by sensors, and last, but not least, the type of expected faults.

Most of the above aspects need forms of intelligence, either basic or advanced, to learn in a time variant, evolving environment and diagnose fault occurrences. These aspects will also be tackled in next chapters.

1.1 How the Book is Organized

We briefly summarize the main topics addressed in the book to get a quick snapshot of both organization and content. The functional dependency among chapters in terms of content is given in Fig. 1.1.

1.1.1 From Metrology to Digital Data

Most embedded systems take advantage of a sensor platform to carry out the due task. However, the mounted sensors may not only be those requested by the application as seen by the user, since extra sensors are generally envisaged to improve the QoS needed by the application or mitigate technological problems that might impair overall performance. As an example of the former class we have the Received Signal Strength Indicator (RSSI) sensor that measures, in wireless communications, the power of the received radio signal. By exploiting the information provided by the RSSI, we can derive the quality of the communication link and identify suitable actions for maximizing it. As an instance of the latter class we have the temperature sensor internal to the sensor device (not to be confused with the temperature sensor mounted on the acquisition board), which measures the temperature of the sensor for thermal effect compensation. In fact, if a tiltmeter is deployed to inspect the structural health of a building, then both the transduction mechanism and the analog to digital conversion are parasitically affected by the temperature. The temperature

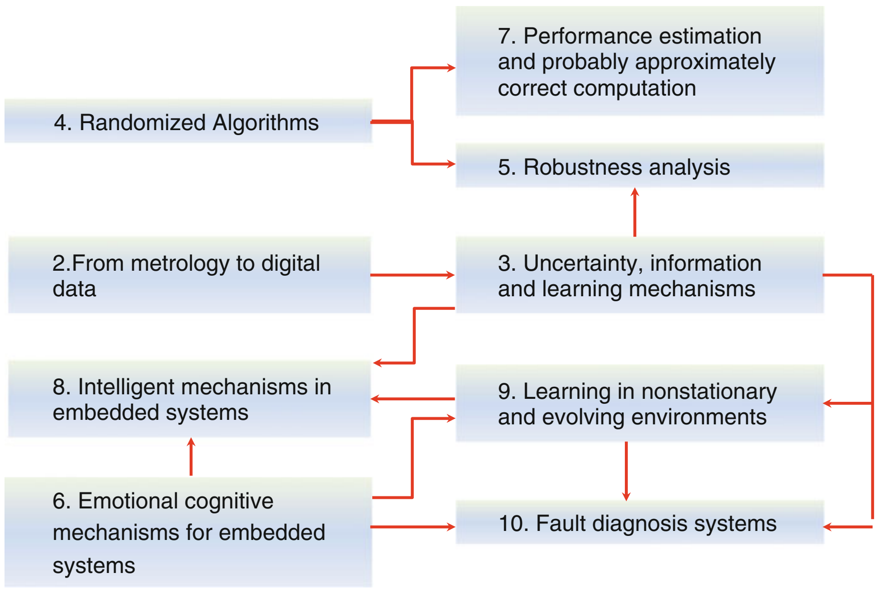
7. Performance estimation and probably approximately correct computation
4. Randomized Algorithms
5. Robustness analysis
2.From metrology to digital data
3. Uncertainty, information and learning mechanisms
8. Intelligent mechanisms in embedded systems
9. Learning in nonstationary and evolving environments
6. Emotional cognitive mechanisms for embedded systems
10. Fault diagnosis systems

Fig. 1.1 The functional dependency among the chapters composing the book. An oriented arc from chapter i to j implies a strong functional dependency between the two chapters with material in i relevant to fully understand that in jth

sensor allows us for introducing measurement compensation actions. As a result, the data flow is affected by uncertainties from the transduction mechanism to the digital instance.

The chapter introduces the basic concepts behind measure and measurements, e.g., accuracy, resolution, and precision and sheds light on the elements composing the measurement chain (transducer, conditioning stage, analog to digital converter, estimation module). Since measurements are affected by uncertainty, we need to investigate how uncertainty corrupts the final acquired data. This analysis sets the basis for the subsequent propagation of uncertainty within the computational chain as well as introduces constraints on the final embedded solution.

1.1.2 Uncertainty, Information, and Learning Mechanisms

Uncertainty associated with available measurements is not the unique form of information corruption mechanism. In fact, in digital embedded systems, finite precision representations introduce an additional form of uncertainty that combines in a nonlinear way with that of measurements and propagates along the computational flow. The outcome is that the information content of the output is corrupted, hence affecting its validity in a subsequent decision-making process.

The chapter introduces and formalizes the most important forms of uncertainty that embedded systems have to deal with. In particular, in addition to measurement uncertainty, we will encounter uncertainty affecting the data representation level in a digital device, characterize it, and see how it propagates within a computational flow.

Another interesting form of uncertainty arises when the computational code in execution on the embedded system contains parametric models, with parameters estimated from available data or parameter-free models where the model is configured directly from available measurements. Machine learning-based solutions represent a relevant example of such mechanisms.

Since machine learning solutions play a main role in intelligent systems, the chapter presents a statistical formalization of the theory of learning by detailing those key points that combine model complexity with uncertainty and accuracy. In particular, we will see that different forms of uncertainty reside behind the learning mechanisms, depending on the noise affecting the data, the effectiveness of the learning algorithm, the number of available instances used by the learning algorithm, and the suitability of the model family envisaged to model the data.

Finally, we introduce uncertainty at the application level either when building a model from incoming data or designing an application solution, which, for its nature, is mostly unknown. Uncertainty at application level means that we can design different solutions for an application, possibly equivalent in terms of performance according to a given figure of merit (i.e., a quantity considered to assess the performance of a system solution or method). The problem of deciding which solution should be preferred among the set of feasible solutions is left to the performance estimation and probably approximately correct computation chapter. Here, differently, we are interested in discovering that a high level source of uncertainty exists and, in many cases, dominates over any attempt to improve performance.

As a final comment, all these sources of uncertainty combine in a nonlinear way and influence the result of the computation in execution on the embedded system. Results are no more uncertainty-free but uncertainty-affected since the embedded system does no more provide a correct deterministic outcome but an uncertainty-affected one.

1.1.3 Randomized Algorithms

The chapter introduces an intuitive key mechanism every engineer should be aware of: randomization. The idea behind the method is very intuitive. Every time we cannot solve a complex problem, either because it is too complex or computationally hard, we explore how the problem behaves on a number of instances. We take an input instance and feed it to the algorithm associated with the problem that provides a result. Then, we repeat the procedure by sampling other instances.

We find it natural to believe that, by having many instances or samples, we should be able to say something about the original problem. This is what the Monte Carlo method is about. However, randomization does much more than a blind sampling from an instance space. In fact, by merging the Monte Carlo method with the probabilistic theory of learning we derive the number of samples needed to solve a large class of hard problems within a probabilistic framework.

The proposed method is general and addresses the very large class of applications and figures of merit that are Lebesgue measurable (i.e., all those involved in a numerical computation associated with both physics and engineering problems). Examples of problems we wish to address in the embedded system world are the assessment of the performance level attained by the embedded application, the evaluation of the energy consumed by an algorithm, the estimation of the time needed to execute a task, the determination of the average latency in providing a result, and the satisfaction of the real-time execution constraint for the application.

After having introduced the main results coming from the randomization theory and those granting convergence of sampled quantities to the exact ones, the chapter will present some general methods based on randomized algorithms for solving a large class of performance assessment problems. Results, independent from the dimension of the sampling space and the probability density function induced on the sampling space, mostly unknown, hold in probability at arbitrary accuracy and confidence levels function of the envisaged number of samples.

1.1.4 Robustness Analysis

Robustness analysis deals with the problem of investigating whether a given solution (e.g., the computational flow in execution on the embedded system) is able to tolerate the presence of uncertainty/perturbations affecting it or not. Many embedded applications rely on subsystems implemented with analog solutions. Which is the impact of the production process on our components and, ultimately, the performance of the system?

If we design a computational solution on a high resolution platform, e.g., data are represented in floating point or double precision and, then, we wish to move the application to an embedded system characterized by a fixed point notation, can we be reassured that performance is within an application-tolerable loss or not? The answer to the questions above is that, a priori, nothing can be asserted unless a robustness analysis is carried out to evaluate/estimate the loss in performance associated with the introduced perturbation effect. Those of us senior enough to have worked on artificial neural networks and having tried to port families of neural models from a high precision platform to a lower precision embedded one are well aware of the drastic performance loss in accuracy we shall expect. Clearly, the problem is well beyond that of a specific application.

Most available results on robustness analysis assume the small perturbation hypothesis to make the mathematics amenable enough so as to derive the closed form relationship between perturbations affecting the computational flow/variables and the induced loss in performance. However, such results prove to be of limited use when envisaging the porting of an algorithm to an embedded system, since the small perturbation hypothesis is mostly violated. The chapter solves this problem by introducing a perturbation in the large analysis that, based on randomized algorithms, allow us to derive an estimate of the robustness index possessed by the application.

1.1.5 Emotional Cognitive Mechanisms for Embedded Systems

The chapter introduces the basis of emotional cognition since it is strongly believed that the next generation of embedded systems will natively integrate such mechanisms either in hardware or in software to allow the device/application to expose sophisticated intelligent behaviors. The focus is on the fundamental cognitive mechanisms obtained by modeling the functionalities of the human brain w.r.t its emotional processing ability. All methods presented in the book following this chapter inherit or expose levels of intelligence that can be associated with a same form of intelligence, either automatic or conscious. For instance, adaptation represents the lowest level of intelligence, finding in the Amygdala its neurophysiology counterpart. Reaction to the input stimuli is ruled by an uncontrolled process that grants an immediate reduced-latency action, modelable as an emotional response to perception. However, in many applications, the use of adaptation mechanisms is not enough, mostly due to the generation of erroneous actions (reactions are based on conservative principles), that must be subsequently validated by higher cognitive levels where conscious controlled processes are activated. For instance, this role is played in our brain by the vertical-medial orbital cortices. The same processes will play a major role in the learning in a nonstationary and evolving environment chapter.

1.1.6 Performance Estimation and Probably Approximately Correct Computation

The PACC theory formalizes the way a computation is carried out within an uncertainty affected environment. As such, it represents the natural characterization of those numerical algorithms for embedded systems or parts of the algorithm affected by those forms of uncertainty presented in Chap. 2. We comment that a deterministic computation, mostly requested to satisfy the worst-case scenario, is generally unacceptable since the cost necessary to grant a deterministic outcome is not justified by the high complexity requested by the solution.

It is shown that, by relaxing the request for determinism, which imposes the application output to be deterministically correct, we can formalize a simpler dual probabilistic framework requesting the computation to be correct in probability. The probabilistic problem is characterized by a lower complexity compared with the deterministic one.

The idea behind PACC—but not its formulation—comes from the robust control community where it has been pointed out that designing a deterministic controller introduces an unnecessary complexity compared to a probabilistic design. This additional complexity is not counterbalanced in most real applications by the gained determinism.

We now recall that a probabilistic computation is the natural way an embedded system processes numerical information since the different forms of uncertainty affecting the computational flow natively provide an output that is correct in probability under some mild hypotheses. Let us provide a simple example. Assume we have a scalar function $f(x)$ whose deterministic output y for each input, spanning its input dominion X, is

$$y = f(x), \forall x \in X.$$

If $f(x)$ needs to be executed on an embedded system, given the presence of uncertainty corrupting its evolution, we are satisfied if

$$\Pr(y \simeq f(x)) \geq \eta, \forall x \in X$$

where η is a confidence term we expect to be close to 1 and $\simeq$ is the approximate operator. In other words, we are satisfied if our embedded system is providing an output approximating the correct one according to a suitable figure of merit; however, the statement has to hold with high probability to be sure that the device behaves as expected, at least in probability. This is exactly what embedded systems, e.g., those designed for domestic appliances, do.

The framework should not be confused with fuzzy logic and fuzzy algorithms that can be cast in the PACC framework but do not naturally provide the confidence level η unless PACC is activated. Randomized algorithms are here used to solve the complex problem associated with the characterization of the PACC level of function $f(x)$.

Another strictly related problem is that of performance estimation and assessment. How can we assess the performance, e.g., in terms of accuracy, of the computation being executed in the embedded system? How can we address the situation where only a given finite dataset is available to estimate the quality of the performance our embedded application claimed to have? If the embedded application is claimed to be 95 % accurate, which is the confidence associated with the statement? The chapter provides answers also to these questions.

1.1.7 Intelligent Mechanisms in Embedded Systems

Adaptation mechanisms are related to those automatic processes implemented by the amygdala and thus allowing our brain to make quick decisions without the need to activate conscious controlled processes. The chapter focuses on some examples of adaptation and conscious decision-making that intelligent embedded systems should possess depending on the functional constraints the application is requesting. At the lowest abstraction level, we have those forms of adaptation affecting the voltage/clock frequency of the system, strategies mainly introduced to keep under control the power consumed by the embedded system. Then, we encounter adaptation at the acquisition level, again with the aim of reducing the energy required to carry out data sampling; the issue is particularly relevant in energy-eager sensors. Here, adaptation basically intervenes on the sampling frequency to reduce the energy consumption.

Intelligence plays also a fundamental role in maximizing the energy harvested when the embedded system can scavenge it from the environment as well as in adapting the system clock to have it aligned with those of neighbors' units. Intelligent mechanisms are beneficial to localize the sensor's unit within an environment without requesting a GPS sensor. In order to achieve this goal, other communicating units have to be deployed nearby and collaborate in a coordinated fashion to the localization effort.

Functional reprogrammability is another form of intelligence that allows the embedded application to undergo changes whenever needed. Although this mechanism is mostly carried out at the software level, with code updated remotely as needed, advances in hardware makes available FPGA-based technologies where reprogrammability can be also envisaged at the hardware level.

1.1.8 Learning in Nonstationary and Evolving Environments

A chapter on "learning in nonstationary and evolving environments" is particularly timely and addresses the case, rather frequent in the real world, where the environment changes but our embedded application does not (it was configured by assuming that the environment was time invariant).

The implications of this way of thinking are fruitful. Before releasing the embedded system, we should ask ourselves if the application we have designed is assuming that the environment and the interaction between the device and the external world will change over time or not. Since all physical processes are, at least, subject to aging phenomena and the environment is mostly time variant, unless suitably controlled (and controllable), we should wonder whether during the lifetime of the embedded application a change is expected or not. In case of a positive answer, we should then ask if the change is negligible or will significantly affect the performance of the embedded application. If this is the case, then the application must be revisited to make it able to deal with changes in the environment or intervene to mitigate the effects of this change.

The main methodologies that allow our application to learn in a nonstationary/time variant environment are presented and detailed. This chapter is fundamental if we have to address the big data scenario where the embedded system might be used to extract features and take a first level decision within a hierarchical triggering mechanism (the embedded system quickly detects events and relevant instances and activates an alarm so that more sophisticated and complex agents intervene to accept/reject the hypothesis). Cognitive mechanisms will play a key role since adaptation itself might not be sufficient to grant the performance level the system is expected to have.

1.1.9 Fault Diagnosis Systems

The last section of the book focuses on fault diagnosis systems. In particular, we will investigate the issue of fault tolerance for sensors and how an application can build mechanisms to detect the occurrence of faults. Here, a cognitive approach will be used, since we want to push the difficult and real case where little a priori information is available and changes and fault signatures must be learned along with the fault diagnosis system directly from the data.

It is shown that little can be done at the single sensor level unless strong hypotheses are made. However, the situation is different if the embedded system mounts a rich sensor platform or is inserted in a sensor network. In such case, redundancy in the information content and functional dependencies among sensors can be exploited to classify a change as fault, change in the environment, or inefficiency of the change detection method (model bias).