HAL
open science

Génération automatique de modèle pour l'optimisation énergétique des systèmes cyber-physiques.

Yohann Rioual

To cite this version:

Yohann Rioual. Génération automatique de modèle pour l'optimisation énergétique des systèmes cyber-physiques.. Electronics. Université de Bretagne Sud, 2019. English. NNT : 2019LORIS533 . tel-03134311

HAL Id: tel-03134311

https://theses.hal.science/tel-03134311v1

Submitted on 8 Feb 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE DE DOCTORAT DE

L'UNIVERSITE BRETAGNE SUD
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies
de l'Information et de la Communication
Spécialité : *Electronique*

Par
Yohann RIOUAL

RL-based Energy Management for Autonomous Cyber Physical Systems

Thèse présentée et soutenue à Lorient, le 9 Décembre 2019
Unité de recherche : UMR6285 – Lab STICC
Thèse N° : 533

Rapporteurs avant soutenance :

Patrick Girard	Directeur de Recherche CNRS, LIRMM Montpellier
Alain Pegatoquet	Maître de Conférence (HDR), Université de Nice

Composition du Jury :

Président :		
Examinateurs :	Patrick Girard	Directeur de Recherche CNRS, LIRMM Montpellier
	Alain Pegatoquet	Maître de Conférence (HDR), Université de Nice
	Daniel Chillet	Professeur, Université de Rennes 1
	Anca Molnos	Ingénieur de Recherche, CEA
Dir. de thèse :	Johann Laurent	Maître de Conférence (HDR), Université de Bretagne Sud
Co-dir. de thèse :	Jean-Philippe Diguet	Directeur de Recherche CNRS, LabSTICC Lorient

Invité(s)

Yannick Le Moullec	Professeur, TalTech, Tallinn, Estonie

Acknowledgement

I would like to thank Patrick Girard, CNRS Research Director at LIRMM laboratory, and Alain Pegatoquet, Associate Professor at University of Nice-Sophia Antipolis for having agreed to be the thesis rapporteurs for this work.

I would also like to thank Daniel Chillet and Anca Molnos, respectively Professor at the University of Rennes 1 and Research Engineer at the CEA for their participation in the jury.

Moreover, I would particularly like to thank my PhD supervisor, Johann Laurent and Jean-Philippe Diguet for their guidance, end-less support, always have belief in my capabilities, understanding and patience.

I had the chance to spent three month working at Taltech, Tallinn, and I would like to thank Pr. Yannick Le Moullec for giving me this opportunity and hosting me during these three months, and also for its expert eye and good advices.

I would also like to thank all my colleagues of Lab-STICC research laboratory at UBS for creating such a pleasant work environment and for being there for me. I would also like to pay special thanks to the administrative assistants, Florence and Virginie, for their support.

I can't forget the friends who accompanied me on this journey. Thus, I would like to thank Alix, Alex, Flaurent, Maxime, Gabriel, Delphine, Erwan, Alexandre K., Mathilde T., Antoine L., Tatiana, Albane, Antoine M., Julien, Clementyne, Hugues, Adrien, Mathilde H., Théo, Guillaume, Stephen, Paul, Pierre, Flavien, Titouan, Mourad, Hugo, Nico and I'm still forgetting people as I write these lines.

And finally, I thank my family whose support has been very helpful throughout my entire life.

"Do what I do. Hold tight and pretend it's a plan!"

The Doctor

Table of Contents

Introduction **6**

1 Theoretical Background **15**

1.1 Introduction to Reinforcement Learning 16
1.2 Markov Decision Processes 17
1.3 Learning a Behaviour Policy 17
1.4 Reinforcement Learning 19
1.5 Neural Networks 20
1.5.1 Description of a formal neuron 21
1.5.2 Artificial neural networks overview 22

2 Related work **25**

2.1 Application schemes of RL approaches 26
2.2 Storing the knowledge in a look-up table 28
2.2.1 Power management 28
2.2.2 Routing protocols optimisation 32
2.2.3 Communications optimisation 35
2.2.4 RL and energy harvesting 39
2.3 RL algorithms with traces 40
2.3.1 Energy management with RL based on traces 41
2.4 Neural network approaches in RL 44
2.4.1 Energy management using neural network approaches in RL 44
2.4.2 Communication optimisation 46
2.5 Others approaches used in RL 49
2.6 Performance improvement in embedded systems thanks to RL 50

3 Energy Management with Reinforcement Learning **53**

3.1 Monitoring marine environment: a case study 54
3.2 Presentation of the selected Markov Decision Process 55
3.3 Look-up table approach with Q-learning 57

3.4 Dyna Q-learning 59
3.5 Neural network approach: Deep Q-learning 61
3.6 Comparative results 64

4 Reward Function Design 69
4.1 Reward Function Evaluation 70
4.1.1 Presentation of the use case 70
4.1.2 Presentation of the decision process 72
4.1.3 Experimental Results 73
4.2 Design of a Piecewise Reward Function 77
4.3 Continuous Reward Function to Balance the Performance and the Energy Consumption 80

5 Multi-agent Reinforcement Learning Approach 87
5.1 Introduction to multi-agent systems 88
5.1.1 Multi-agent learning 90
5.1.2 Hysteretic Q-Learning 91
5.2 Decentralized energy management 91
5.2.1 Independent Management of Sensors 93
5.2.2 Results of the proposed algorithm with the marine buoy use case . 94

Conclusions and Perspectives 101

Abbreviations 116

Nomenclature 117

Personal Publications 125

Introduction

A Cyber-Physical System (CPS) is a computer-based system or a network interacting with physical processes. Their potential applications include: intervention (e.g., collision avoidance); precision (e.g., robotic surgery and nano-technology manufacturing); operation in dangerous or inaccessible environments (e.g., search and rescue, fire fighting and abyssal sea exploration); coordination (e.g., air traffic control, war); efficiency (e.g. net zero energy buildings); and improvement of human capabilities (e.g., health monitoring). Examples therefore are shown in Fig. 1.

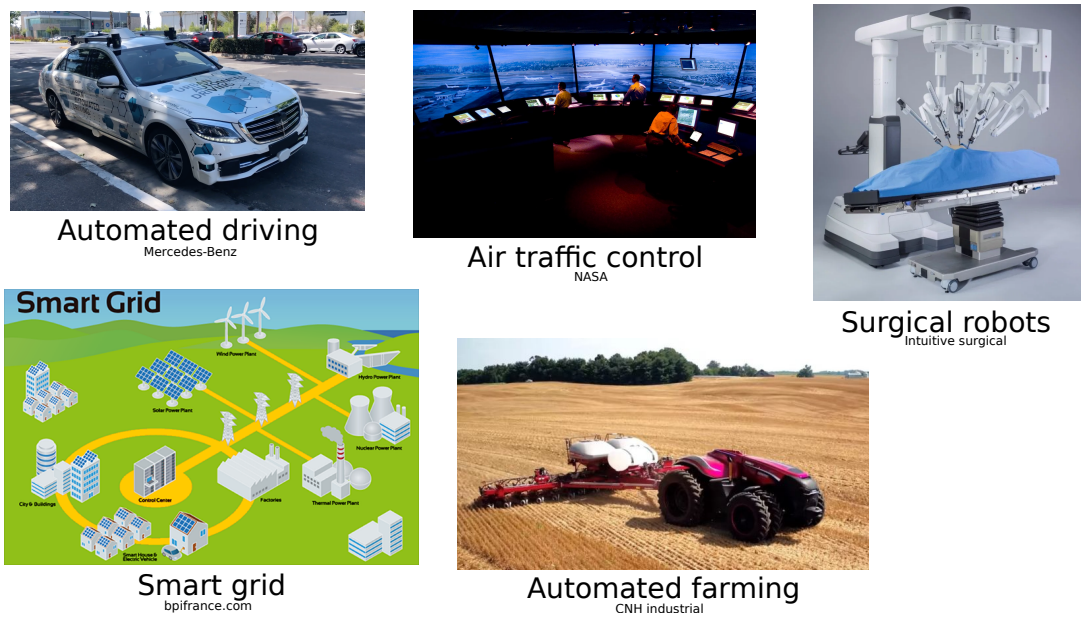

FIGURE 1 – Examples of different cyber-physical system applications

One application of CPS is the remote monitoring of complex physical and biological phenomena. This type of application is growing rapidly thanks to recent advances in Internet-of-Things (IoT) paradigms. There is a lot of interest for IoT nodes; between 2015 and 2025, the number of IoT connected devices installed is expected to increase by 489% (Fig. 2). Indeed, IoT nodes are capable of collecting and transmitting data autonomously. The position of these nodes is not necessarily predetermined, and they can be connected in a mesh network within which they communicate. Thus, IoT nodes

are becoming more complex in order to meet increasing needs for accurate environmental observations. Nevertheless, the limited computing, energy and memory resources of the IoT devices restrict their deployments.

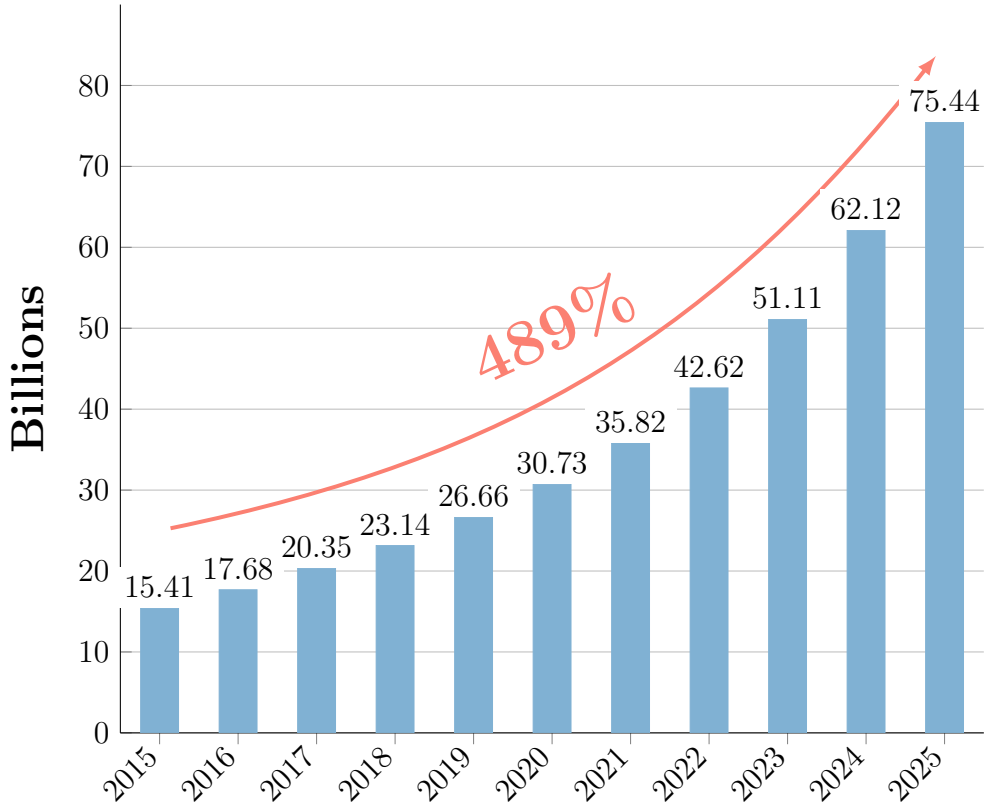

FIGURE 2 – IoT connected devices installed base worldwide from 2015 to 2025 (in billions) (source IHS [1])

An autonomous IoT node is composed of different modules (Fig. 3): a processing unit, a communication unit, one or more sensors and a battery. A major limitation to the deployment of these nodes is the limited amount of energy. IoT nodes need to be small to be used in various environments and the capacity of the battery is limited. Battery technology improves significantly slower than other electronic parts of a node [2]. Energy harvesting is a promising technique that extends battery lifetime and provides a satisfactory quality of experience for IoT devices ([3], [4], [5]). The energy harvesting module enables an IoT device to capture the ambient renewable energy such as solar radiation, wind power generation, radio-frequency signals, or kinetic human motion to supply the energy for the IoT consuming tasks (Fig. 4). These tasks include sensing, processing and communicating.

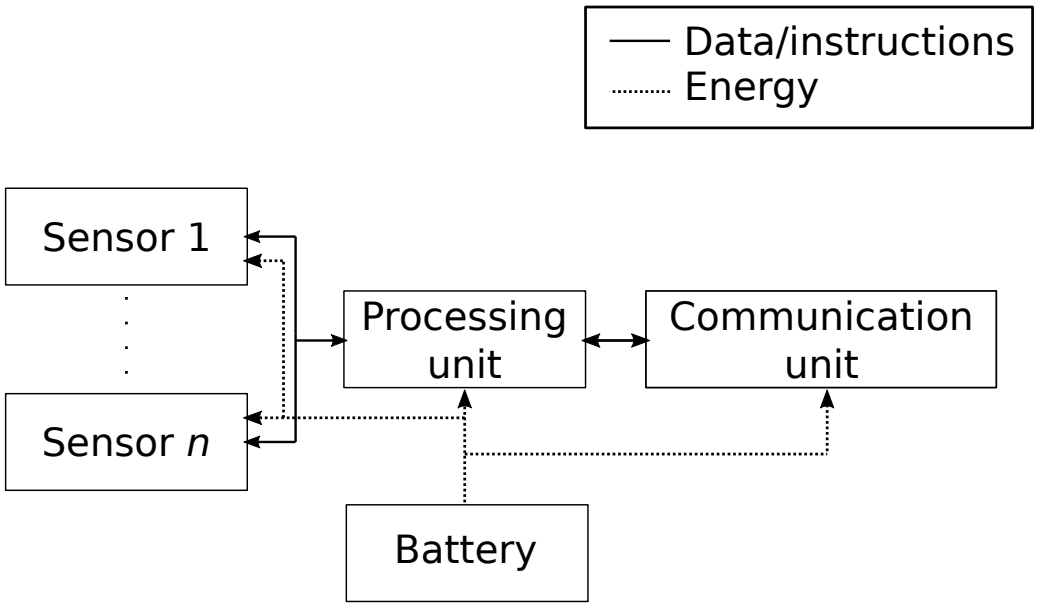

FIGURE 3 – Schema of an IoT node

Among all technologies of energy harvesting, the solar panel technology is the most efficient way to harvest energy (up to 15 mW/cm^2) (Table 1). Nevertheless, the harvesting capability of a solar panel depends on the ageing of the components [6] and the weather (sun irradiation, sky cover). Therefore, the harvested energy greatly varies over the day, and increases the uncertainty in the availability of energy resources of such systems.

Harvesting technologies	Power density
Solar cell (outdoors at noon)	15 mW/cm^2
Wind flow (at 5 m/s)	16.2 µW/cm^3
Vibration (Piezoelectric – shoe insert)	330 µW/cm^3
Vibration (electromagnetic conversion at 52 Hz)	306 µW/cm^3
Thermoelectric (5 °C gradient)	40 µW/cm^3
Acoustic noise (100dB)	960 nW/cm^3

TABLE 1 – Power density of energy harvesting technologies

Furthermore, the energy consumption of an IoT node is difficult to model, the uncertainties increase with the complexity of the micro-controller hardware and the use of an Operating System (OS). Indeed, micro-controllers are more and more powerful; to achieve such performance, they took advantages of processor hardware evolutions such as cache memory, branch prediction, instruction pipelining. The improvement in perfor-

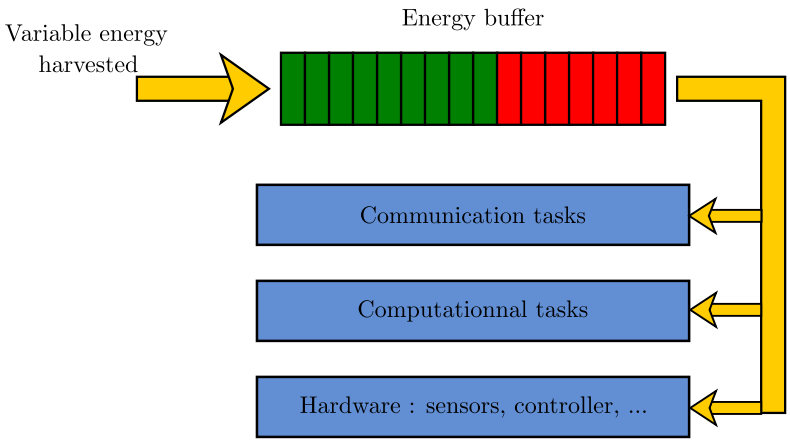

FIGURE 4 – Energy consuming tasks of a node

mance results in non-deterministic energetic behaviour. It becomes hard to model and to predict the energy consumption of such architectures. Moreover, some systems have heterogeneous computing architecture such as ARM big.LITTLE, which couples relatively battery-saving and slower processor cores (LITTLE) with relatively more powerful and power-hungry ones (big) or hardware accelerator along the general processor which makes their architectures more complex.

In addition to an increasingly complex architecture, many CPS use an OS. This eases the development of an effective application software, providing a uniform framework for organizing and accessing the software and hardware resources. With an OS, applications are organized as a collection of independent threads of execution. The OS decides which thread should be executing by examining the priority assigned to each thread by the application designer. When an interrupt occurs, if its priority is higher than the current task, the OS temporarily interrupts the task without requiring its cooperation to run another task with a higher priority. To preserve the energy of the system, the OS allows the processor to spend more time in a low power mode. A wake up signal get the system out of the low power mode. The occurrence of the wake up signal is unpredictable and add uncertainty to the energy consumption. A drawback for the use of an OS is that embedded systems have limited amount of memory and processing capabilities and the OS uses a part of the available memory and processing time to run.

The OS makes application development more flexible, i.e., it allows the designer to focus on application development rather than resource management. And at the same time, it increases the uncertainties on the behaviour of the node making unpredictable

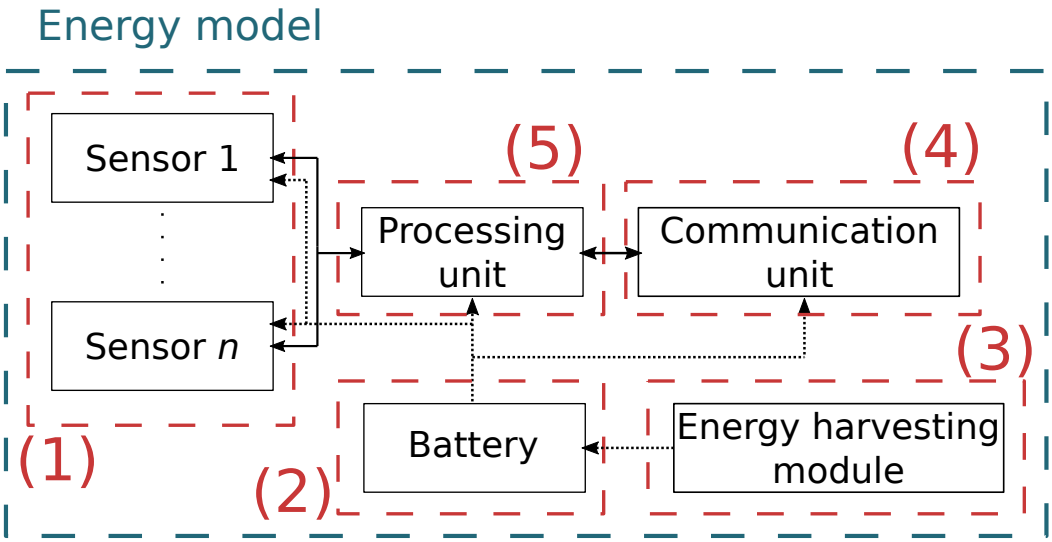

FIGURE 5 – The design of an energy model for an IoT node faces many hurdles

the energy consumption. Building an energy model for an embedded system is a difficult task when no OS is involved, but when an OS is added and the node energy relies on energy harvesting, an accurate energy model becomes infeasible.

The energy management of an IoT node is usually done using an energy model designed a priori in a laboratory. Nevertheless, the ageing of the components and the uncertainties further complicates the design of an accurate energy model (Fig. 5). Each module of the node is ageing and lost in energy efficiency over time. The sensors lose accuracy, wear and tear of mechanical part increases and their energy consumption increases too. The battery (2) capacity decreases over time due to different stress factors such as the temperature, the depth of discharge, the charge current and the discharge current. As the energy harvesting module ages (3), it harvests less energy due to external atmospheric conditions to which it is exposed (sun, wind, particles, rain or even snow, etc). The other type of factors limiting the lifetime of the harvesting module are internal and related to the quality of the materials used; for a solar panel it includes the quality of the semiconductors on which are based unit solar cells. In this case too, there are ageing effects which are mainly related to the influence of external conditions such as temperature. The communication unit (4) increases its energy consumption with the ageing process [7], moreover the energy consumption of the radio depends of the communication channel state (the transmission power is higher in a noisy channel or the transmission must repeated if the data are lost). Moreover, the energy consumption changes depending to the uncertainties created by the

architecture or the OS.

Thus, measurements are required to produce optimal and reliable model, highly detailed specific material data and information about the different module of the node works, ages and how they influence each other. Furthermore, their energy consumption evolves according to environmental conditions that are numerous and so, hard to replicate. Finally, one can come to the conclusion that the model must be built and adapted on-line, during the node deployment. In this thesis, we address the problem of energy management. Considering the unavailability of an accurate energy model, we proposed a solution where the IoT node is considered as a black box. It turns the problem into a search of learning and adaptation method in order to adapt at runtime the node to its environment.

Reinforcement learning is a type of machine learning that can handle uncertainties and so is a promising candidate method to provide the sensor nodes with the ability to adapt according to the available energy. Nevertheless, the use of such approaches faces many challenges. Many approaches exist but only some of them are suitable to be used in embedded systems. Moreover, reinforcement learning algorithms have many parameters to tune and rely on expertise from the designer. Thus, the objective of this thesis is to provide designers with guidelines to help them use reinforcement learning approaches for the energy management of autonomous cyber-physical systems.

Thesis contributions

The four main contributions of this thesis are the following:

- *Proposition of metrics to select the appropriate algorithm depending on a given application*: The selection of the appropriate algorithm for a given application is challenging. Various algorithms exist and all of them are not suitable for a use in embedded systems. Indeed, embedded systems have limited memory and processing capabilities and some algorithms will require more than what the system can provide. Furthermore, there is a lack of guidelines for designers to select the appropriate approach. Thus, the first contribution of this thesis is to compare different approaches to define metrics to help designers choose the approach according to their application and system.
- *Highlighting of variables and parameters influencing the performance of a reward function*: Once the selection of the appropriate algorithm, the designer needs to design a reward function which will give the correct behaviour to the node. Ho-

wever, the literature rarely discusses the choice of the reward function design. We evaluated different reward functions to identify the most suitable variables to consider when designing such a function for the energy management of a sensor node. And as the second contribution of this thesis, we have explored how to design an efficient reward function.

- *Proposition of two reward functions adjusting the performance according to the battery charge level*: We use our second contribution to propose as a third contribution to link two opposite objectives in a reward function. Indeed, most applications of IoT must do a trade-off between a performance criterion and the energy consumption. Thus, we present two reward functions able to adjust automatically the performance of a sensor node according to its battery charge level.
- *Proposition of a multi-agent reinforcement learning algorithm able to control, independently, sensors with different energy consumptions*: The use of a single agent to learn the energy management of a node shows some limitations; to independently control the different sensors, it is necessary to add the actions in the set of action of the decision process, which increases the size of the look-up table and makes it ineffective. So, the fourth contribution of this thesis is the use of multi-agent reinforcement learning and an algorithm able to control independently the measurement frequency of several sensors according to their respective energy consumption.

Outline

The remaining sections of this thesis are structured in 6 chapters as follows:

- **Chapter 1: Theoretical Background:** This chapter provides the necessary theoretical knowledge required for understanding the reinforcement learning. In this chapter, the single-agent reinforcement learning approach is described, as well as the functioning of neural networks.
- **Chapter 2: Related Work:** This chapter provides an overview of the state of the art on the use of reinforcement learning approaches for the energy management of embedded devices.
- **Chapter 3: Energy Management with Reinforcement Learning:** This chapter presents a comparison of different approaches for the energy management for a IoT node. Different metrics provide guidelines to a designer to select the appropriate approach in function of the application and system capabilities.

- **Chapter 4: Reward Function Design:** The chapter presents a comparison of different reward functions to find the most interesting features to use when designing a reward function. With these results, two reward functions are proposed and presented. These reward functions adjust the measurement frequency of each sensor according to the battery charge level.
- **Chapter 5: Multi Agent Reinforcement Learning:** In Chapter 5, multi-agent reinforcement learning is used to manage the energy consumption of a sensor node according to its battery charge level. An algorithm is proposed to control independently the different sensors according to their respective energy consumption.
- **Conclusion and Perspectives:** In the last chapter, we summarize all the works presented in this thesis and presents perspectives opened by our work.

Chapter 1

Theoretical Background

Contents

1.1 Introduction to Reinforcement Learning 16

1.2 Markov Decision Processes 17

1.3 Learning a Behaviour Policy 17

1.4 Reinforcement Learning 19

1.5 Neural Networks 20

1.5.1 Description of a formal neuron 21

1.5.2 Artificial neural networks overview 22

The interest in Reinforcement Learning (RL) has arisen with the success in various domains such as Atari game [8], robotic control [9], traffic light control [10] or energy management [11]. Nevertheless, a multitude of reinforcement algorithms exists and not all of them are suitable for an use on embedded systems such as sensor node. Indeed, the needs in computation power and memory are high. There are two main approaches used, the first one which stores the knowledge in a table and the other one which uses approximators to computes the value of the different possible actions with the previously learned information. This chapter is an introduction to reinforcement learning and presents the theory. The neural networks are also introduced since they are used as approximators in some reinforcement algorithms used in this thesis.

1.1 Introduction to Reinforcement Learning

RL [12] is a formal framework that models the problem of sequential decisions, in which an agent learns how to take better decisions by interacting with its environment (Fig. 1.1). When the agent performs an action, it receives as a feedback the new state of its environment and a reward signal, encoding the information on the quality of the transition. The agent's objective is to maximize its reward in the long-term.

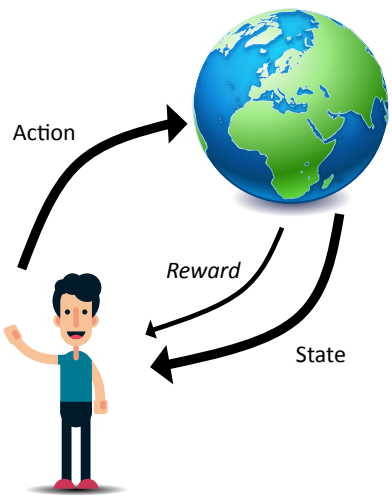

FIGURE 1.1 – Interaction agent-environment

The following sections present the mathematics that explains how an agent is able to learn to make decisions in a dynamic environment.

1.2 Markov Decision Processes

Traditional single-agent reinforcement learning is modelled as a discrete-time, finite Markov Decision Process (MDP). MDPs have been commonly used for solving sequential decision making problems where the agent also has to take into account the dynamics of the environment. An MDP is defined as a 4-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ where:

- $\mathcal{S}$ is a state space;
- $\mathcal{A}$ is a set of actions;
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1[$ is a *transition function* specifying, for each state, action, and next state, the probability of that next state occurring;
- $\mathcal{R}$ is a *reward function*, specifying, for each state, action, and next state, the expected immediate reward.

At every time-step t, the agent observes the current state of its environment, $s_t \in \mathcal{S}$, and chooses a corresponding action, $a_t \in \mathcal{A}$, to perform. After completing its action, the environment moves to the next state, $s_{t+1} \in \mathcal{S}$, given the transition probability $\mathcal{T}(s_t, a_t, s_{t+1})$, and the agent receives the reward signal r_t according to the reward function $\mathcal{R}(s, a)$. Figure 1.1, illustrates the explained interaction cycle that establishes the foundation for reinforcement learning.

In this thesis, MDPs are assumed to be stationary, i.e. the elements of the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ do not change over the time.

1.3 Learning a Behaviour Policy

As the agent experiences these interactions, it gradually learns how to map the states to the actions, as a form of a behaviour *policy* $\pi : \mathcal{S} \rightarrow \mathcal{A}$, such that the largest long-term pay-off is obtained. The accumulated discounted reward signals that the agent receives by performing its actions from an arbitrary state s according to a policy $\pi(s)$ is referred to as the *state-value* function of the policy $V^\pi(s)$. Thus, for every policy π, $V^\pi(s)$ can be calculated as:

$$V^\pi(s) = \mathbb{E}_\pi \left(\sum_{t=0}^{\infty} \gamma^t r_t \mid s_t = s \right) \tag{1.1}$$

The discount factor $\gamma \in [0, 1]$ indicates the importance of the long-term cumulative rewards over the short-term pay-off of the actions. A discount factor of 0 makes the agent myopic by considering only the immediate rewards that the agent obtains after performing every action, whereas a larger factor close to 1 implies more distant rewards. If the discount factor is close to 1, without a terminal state, or if the agent never reaches one, all environment histories become infinitely long, and Q-values with additive, undiscounted rewards generally become infinite.

To satisfy its main objective of maximizing the discounted cumulative reward signals, the agent must learn the optimal policy $\pi^*(s)$ such that:

$$V^{\pi}(s) \leqslant V^*(s), \forall \pi, s \tag{1.2}$$

If the optimal policy is discovered and both the transition probabilities and reward values are known, the value of the optimal policy $V^*(s)$, can be calculated using the *Bellman optimality equation* [13].

$$V^*(s) = \max_{a \in \mathcal{A}} \left[\mathcal{R}(s, a) + \gamma \sum_{s'} \mathcal{T}(s, a, s') V^*(s') \right] \tag{1.3}$$

As an alternative to the state-value function $V^{\pi}(s)$, a *state-action* value function, $Q^{\pi}(s, a)$, can be used for optimization of the agent's behavior. $Q^{\pi}(s, a)$ specifies the sum of discounted rewards that the learning agent expects from following the policy π, after performing action a in state s. Formally referred to as the Q-function, $Q(s, a)$ maps both the states and the actions that can be performed at those states as a pair to the corresponding rewards that the agent expects to receive, $(Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R})$. Thus, similar to the formula shown in 1.1, $Q^{\pi}(s, a)$ denotes:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi} \left(\sum_{t=0}^{\infty} \gamma^t r_t \mid s_t = s, a_t = a \right) \tag{1.4}$$

As mentioned earlier, the primary goal of the agent is to learn the optimal policy π^* among with every policy π, that yields the maximum accumulated long-term reward.

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a) \tag{1.5}$$

Similar to equation (1.3), the Q-function of the optimal policy $Q^*(s, a)$ can be described as Bellman's optimality equation:

$$Q^*(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s'} \mathcal{T}(s, a, s') \max_{a' \in \mathcal{A}} Q^*(s', a') \tag{1.6}$$

Given the recursive description that Bellman's optimality equation (1.6) provides, the agent can utilize techniques such as *dynamic programmings* (DP) to calculate $Q^*(s, a)$ of the optimal policy and update the policy accordingly. In equation (1.6), the agent looks at the Q-value of every action a' in the next state s' to find the action that results in the highest expected $Q^*(s', a')$. Once the maximum $Q^*(s', a')$ is found, the agent can update $Q^*(s, a)$ (1.6) and the $V^*(s)$ (1.7) values accordingly.

$$V^*(s) = \max_a Q^*(s, a) \tag{1.7}$$

Equation (1.8) means that in order to update $V^*(s)$, the agent must find the action a that results in the highest discounted total reward $Q^*(s, a)$ in state s. Therefore, upon updating $Q^*(s, a)$, the agent's policy also gets updated so that it maps the state s to the best action a.

$$\pi^* = \operatorname*{argmax}_a Q^*(s, a) \tag{1.8}$$

1.4 Reinforcement Learning

As shown previously, dynamic programming (DP) can recursively calculate the Q-value of the optimal policy $Q^*(s, a)$, and for problems that have small state-action spaces, dynamic programming is considered an efficient approach to compute the optimal policy $\pi^*(s)$ [14]. However, in order to efficiently use the DP technique, knowing both the transition function $\mathcal{T}(s, a, s')$ and the reward function $\mathcal{R}(s, a)$ are required. This is an issue for DP, since in most real-world scenarios, the problems are complex and having prior knowledge of a complete model of the environment and its dynamics that includes both the transition probability and the associated rewards of every state-action pair, is often not possible. The complexity of the problem also means that the state-action space may

be large that would make DP computationally inefficient or even infeasible in case the state-action space is continuous.

Instead of using dynamic programming, the learning agent can gradually find an optimal policy through interactions with the environment without the requirement of knowing the dynamic model of the environment beforehand. One of the most well-known algorithms that is commonly used in reinforcement learning, is called "Q-learning". Q-learning is a "model-free" RL approach that aims at directly finding the optimal policy and learning the Q-function, as opposed to learning the complete dynamic model. In "model-based" reinforcement learning approaches, the agent attempts at learning the complete model by capturing the transition probabilities and reward function. Similar to the case of dynamic programming, model-based approaches may become inefficient when the MDPs have large state-action spaces.

Exploration-exploitation dilemma

A major issue in RL is the dilemma between exploration and exploitation. Exploration chooses an action randomly in the system to find out the utility of that action. Whereas exploitation deals with the actions which have been chosen based on the previously learned relevance of this action. However, acting greedily before the convergence may lead to sub-optimal policies because the agent would not have had the opportunity to sample state-actions pair that might lead to higher returns. In order to avoid this, we follow a method to select an action called ϵ-greedy policy, where the agent chooses the action that it believes has the best long-term pay-off with the probability $1-\epsilon$. ϵ is a tuning parameter, which sometimes changed, either according to a fixed schedule (reduce progressively the exploration), or adaptively based on heuristics.

The exploration finds new interesting actions to converge to an optimal policy. The policy is found by the agent using either a table which stores the Q-values or an Artificial Neural Network (ANN) to compute these values. The following section presents the functioning of a formal neuron and of ANNs .

1.5 Neural Networks

Artificial Neural Networks (ANNs) are a set of algorithms whose design is inspired by the functioning of biological neurons and which are nowadays similar to statistical

methods. They are networks of simple processing elements (called neurons) that are interconnected, calculating on their local data and communicating with the other elements. Their fields of application are varied: statistics (data analysis, forecasting, classification), robotics (control and guidance of robots or autonomous vehicles), pattern recognition, signal processing, learning simulation, ...

In the biological model, neurons receive signals (electric impulses) from other neurons by dendrites and send the information by axons. The contacts between two neurons (between axon and dendrite) are done through the synapses. The signals do not operate in a linear way: threshold effect.

1.5.1 Description of a formal neuron

By analogy with the biological neuron, the formal neuron is a model characterized by an internal state $s \in S$, input signals $x_1, \cdots, x_p$ and an activation function f.

$$s = h(x_1, \cdots, x_p) = f(\alpha_0 + \sum_{j=1}^{p} \alpha_j x_j) \tag{1.9}$$

The activation function transforms an affine combination of the input signals, α_0 being called the neural bias. This affine combination is determined by a weight vector $[\alpha_0, \cdots, \alpha_p]$ associated with each neuron and whose values are estimated during the learning phase. They constitute the "memory" of the network. The different types of neuron are distinguished by the nature of their activation function f. There are many different activation functions and the main functions are :

- linear: f is the identity function;
- sigmoid: $f(x) = \frac{1}{1+e^x}$;
- rectifier: $f(x) = x^+ = \max(0, x)$;
- radial: $f(x) = \sqrt{\frac{1}{2\pi}} e^{\frac{-x^2}{2}}$;
- ...

Linear and sigmoidal models are well adapted to learning algorithms involving gradient back-propagation because their activation function is differentiable; they are the most commonly used. The threshold model is probably more in line with biological "reality" but poses learning problems. Neurons can be pushed into states in which they become inactive. In this state, no gradients flow backward through the neuron, and so the neuron

becomes stuck in a perpetually inactive state and "dies". This is a form of the *vanishing gradient problem*. In rare cases, large numbers of neurons in a network can become stuck in dead states, effectively decreasing the model capacity. This problem typically arises when the learning rate is set too high.

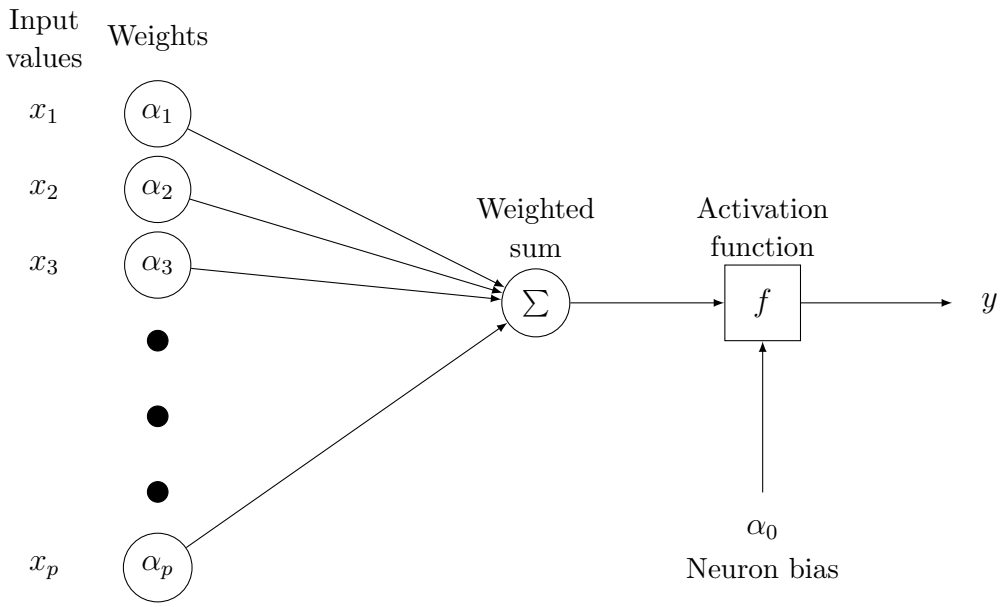

FIGURE 1.2 – Model of a Artificial Neuron

1.5.2 Artificial neural networks overview

Artificial neural networks (ANNs) are statistical models directly inspired by, and partially modelled on biological neural networks. They are capable of modelling and processing non-linear relationships between inputs and outputs in parallel.

ANNs are characterized by containing adaptive weights along paths between neurons that can be tuned by a learning algorithm that learns from observed data in order to improve the model. In addition to the learning algorithm itself, one must choose an appropriate cost function. The cost function is used to learn the optimal solution to the problem being solved. This involves determining the best values for all of the tunable model parameters, with neuron path adaptive weights being the primary target, along with algorithm tuning parameters such as the learning rate. Theses optimization are done by techniques such as gradient descent or stochastic gradient descent. The goal is to make the ANN solution be as close as possible to the optimal solution, which when successful means that the ANN is able to solve the intended problem with high performance.

Architecturally, an artificial neural network is modelled using layers of artificial neurons, or computational units able to receive input and apply an activation function along with a threshold to determine if messages are passed along. In a simple model, the first layer is the input layer, followed by one hidden layer, and lastly by an output layer (Fig. 1.3). Each layer can contain one or several neurons. Models can become increasingly complex, with increased abstraction and problem solving capabilities by increasing the hyperparameters. The hyperparameters express "high-level" properties of the model such as the number of hidden layers, the number of neurons in any given layer, and/or the number of paths between neurons. When the model complexity increases, the chance of overfitting also increases. The overfitting appears when the network extracted some of the residual variation (i.e. the noise) as if that variation represented underlying model structure.

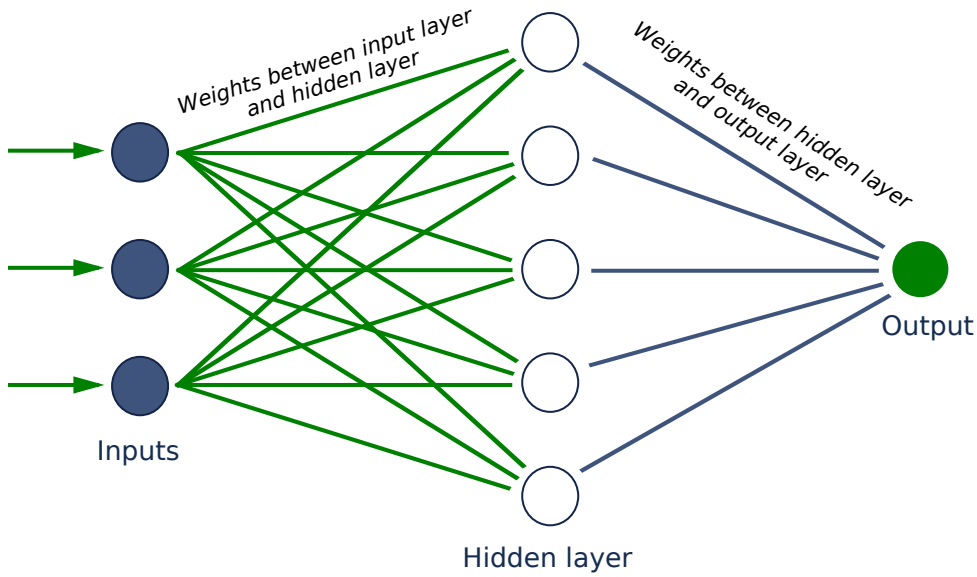

FIGURE 1.3 – Structure example of a multilayer perceptron with a single hidden layer

Model architecture and tuning are therefore major components of ANN techniques, in addition to the actual learning algorithms themselves. All of these characteristics of an ANN can have significant impact on the performance of the model. Additionally, models are characterized and tunable by the activation function used to convert a neuron's weighted input to its output activation. There are many different types of transformations that can be used as the activation function.

The abstraction of the output as a result of the transformations of input data through

neurons and layers is a form of distributed representation, as contrasted with local representation. The meaning represented by a single artificial neuron for example is a form of local representation. The meaning of the entire network however, is a form of distributed representation due to the many transformations across neurons and layers.

One thing worth noting is that while ANNs are extremely powerful, they can also be very complex and are considered as black box algorithms, which means that their inner-workings are very difficult to understand and explain.

The following chapter presents the work done with the use of reinforcement learning to improve the energy management of sensor nodes.

Chapter 2

Related work

Contents

2.1 Application schemes of RL approaches 26

2.2 Storing the knowledge in a look-up table 28

2.2.1 Power management 28

2.2.2 Routing protocols optimisation 32

2.2.3 Communications optimisation 35

2.2.4 RL and energy harvesting 39

2.3 RL algorithms with traces 40

2.3.1 Energy management with RL based on traces 41

2.4 Neural network approaches in RL 44

2.4.1 Energy management using neural network approaches in RL . 44

2.4.2 Communication optimisation 46

2.5 Others approaches used in RL 49

2.6 Performance improvement in embedded systems thanks to RL 50

Reinforcement Learning (RL) enables a new paradigm to solve the energy management problem in embedded systems with harvesting capabilities. Many approaches using RL were proposed in the last years to address the non trivial challenge of designing efficient adaptation algorithms. These approaches must be suitable for the limited resources provided by sensor nodes in terms of memory, computation power, and energy storage. This chapter exposes a comprehensive overview of the state of the art in energy optimization approaches with RL.

As stated in Chapter 1, the goal of an agent in RL is to maximize its cumulative reward by finding the optimal behaviour policy. In some cases, the policy may be a simple function or look-up table, whereas in others it may involve extensive computation such as a neural network. Section 2.1 presents different applications where RL approaches are used in order to optimize the energy consumption. Section 2.2 presents the applications using a look-up table to store the Q-values. Then, Section 2.3 presents the approaches using eligibility traces. The Section 2.4 presents approaches with neural networks to find the best policy. And finally, Section 2.5 presents less common algorithms with different approaches.

2.1 Application schemes of RL approaches

Reinforcement learning is an universal solution to most problems related to the dynamics and uncertainty of the operating environment [12]. Thus, RL has been applied in various schemes:

- *Medium Access Control (MAC) protocols* coordinate channel access among multiple nodes in a single-hop transmission to reduce collisions. Two main functions are sleep-wake scheduler and transceiver selector:
  - *Sleep-wake scheduler* arranges the transmission, reception, idle and sleeping time duration. During the idle mode, sensor nodes listen for potential packet transmissions and the energy consumption is almost identical to the receive mode. To reduce energy consumption, a sleep-wake scheduler schedules sleeping and waking (i.e. transmission, reception and idle) time duration. Longer waking time duration (or higher duty cycle) increases bandwidth availability leading to higher throughput and lower packet latency; however, it also increases energy consumption. The waking time duration increases with the network traffic load or QoS requirements. RL has been applied to optimize the energy consumption

in slot assignment [15] or to adjust the sleeping and waking time durations depending on the data send by the neighbouring [16].

  * *Transceiver selector* selects either a long-range or short-range radio for data and control packet transmissions. Long-range (short-range) radio uses higher (lower) transmission power. To reduce energy consumption, a transceiver selector agent switches between the transceivers based on physical range (e.g. whenever a mobile node moves from one effective transmission range to another) and channel conditions (e.g. fading, interference, shadowing, and multi-path effects) [17].

- *Self learning radio* adapts dynamically the power needed for the efficient transmission of data depending on the quality of the communication channel. The agent reduces the power consumption of the transmission while respecting the quality requirements within the network ([18], [19]).
- *Cooperative networks* where the nodes work together to improve the communication or the energy consumption of the overall network. The cooperation can occur in communication to select cooperative forward packets towards sink nodes in order to reduce the effects of deteriorating channel conditions and changes in network topology [20]. A cooperative network accomplishes an entire team learning of sleep scheduling by rotating the role of active node to preserve the network lifetime using RL in [21].
- *Routing* enables a sensor node to search for the best route to a sink node. The sink node collects data from all nodes in the network through single or multiple hops, and subsequently send them to remote servers. RL has been applied in each sensor node to learn the best route to the sink node ([22], [23]).
- *Rate control* adjusts the packet transmission rate of a source node, and hence the congestion level of intermediate nodes, along a route [24].
- *Task scheduling* schedules and carries out the right task at different time instant. For instance, in [25], RL has been applied in each sensor node to learn the usefulness of each task (i.e. detect targets, track targets, send data about targets, predict trajectory, intersect trajectory and sleeping) at different time instant in order to reduce energy consumption.
- *Power management* adapts the power mode of each node depending on the workload in order to reduce the consumption without degrading the performance. For instance, in [26], the agent selects the appropriate power mode according to the probability that a wake-up signal occurs reducing the energy consumption more

efficiently.

As seen in this section, there are various schemes where RL can solve problems related to uncertainty in the environment. Moreover, there are various RL algorithms which can be used and the following sections give an overview of the most frequently used approaches in the literature.

2.2 Storing the knowledge in a look-up table

In a discrete environment with a discrete actions space, only the corresponding Q-values are needed. In this case, a convenient approach is to store the Q-values in a look-up table. Moreover, the most popular algorithm in the literature is the Q-learning [27] and it uses a look-up table. Thus, in this section, we present a overview of the applications of the look-up table for different applications.

2.2.1 Power management

For the power management problem, different approaches exist to adapt the consumption according to different metrics such as the harvested energy or the workload. Usually, the nodes are deployed for prolonged period with limited resources. Given this, a goal is to minimize wasted energy, especially when the node is idle.

In [26], the authors present a model of adaptive power management of an IoT System-on-Chip (SoC) based on the Q-learning algorithm. The objective is to select the less consuming power mode when the node is waiting for a wake-up signal. This signal has $1-p$ probability of occurring. The state space is composed of the different available power modes and the set of actions (Fig. 2.1) is the possible transition between the system's power states (A_0: Stay in the same state; A_1: Clock switching to 8 MHz; A_2: Clock switching to 32 kHz; A_3: Switch from current state to Sleep; A_4: Switch from current state to DeepSleep). The wake-up signal returns to the Idle-16 MHz state and locks the system until the next suspend sequence. Each action is achieved on a specific number of system cycles depending on the system implementation. The usual default policy is to select the action from the state s_t with the lowest coefficient:

$$a_{t+1} = \underset{a}{\operatorname{argmin}}\, Q_t(s_t, a) \tag{2.1}$$

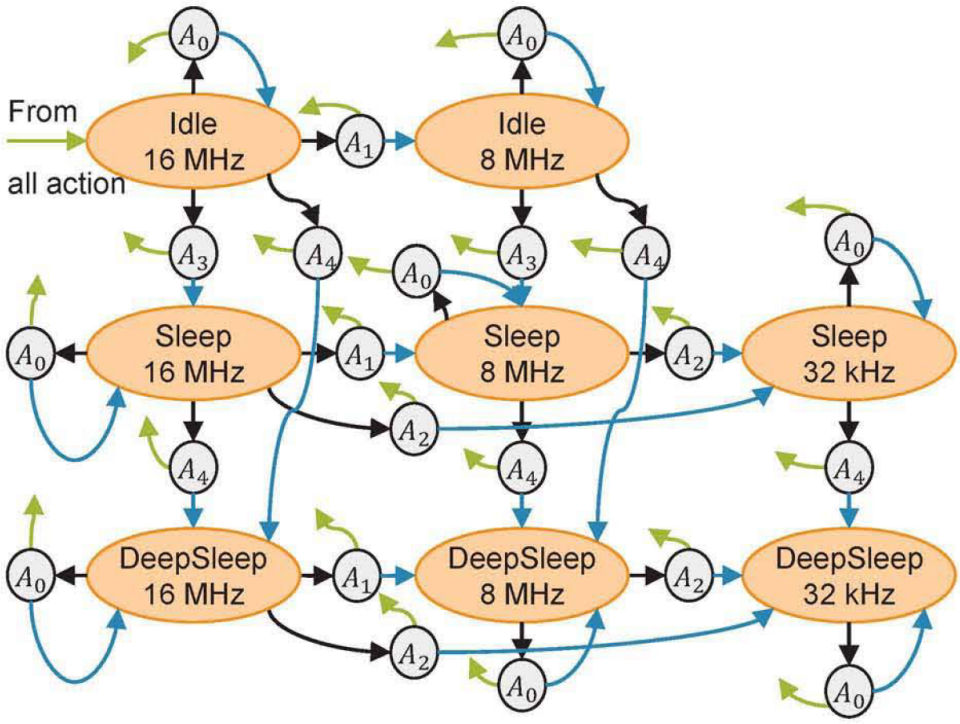

FIGURE 2.1 – Markov Decision Process diagram showing the set of actions and transitions [26]

The Q-value associated to the selected action is updated as shown in Eq. 2.2 :

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(\delta E + \gamma \min_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)) \tag{2.2}$$

where the reward is the energy consumption δE of the system during the state operation. The agent will select the action with the lowest Q-value to minimize the energy consumption. The authors conducted several simulations which shows an average gain of 17% compared to static decisions.

The Q-learning algorithm is also used to adjust the performance and the energy consumption according to the workload. In this way, [28] presents an on-line power management approach. The authors propose to differentiate the energy management of the peripheral devices and the CPU. Indeed, they have different operating behaviours and performance evaluation. Their proposed power manager does not require any prior knowledge to adjust the power consumption depending to the workload.

The simulation results for peripheral devices show that for a low latency expert-

algorithm outperforms Q-learning due to the fact that they are designed for high performance. Whereas Q-learning approach allows the device to buffer the requests. Q-learning outperforms when the performance is relatively less important than the power consumption and it provides wider range of power-performance trade-off. Moreover, in contrast to the expert based policy, the Q-learning power management algorithm not only learns and adapts to different workloads, but also adapt to different hardware. When applied to a microprocessor, the Q-learning based controller can correctly learn the trade-off space and give effective control policies to respect constraint on CPU temperature, power consumption or performance.

The previous work on microprocessor uses a well-known power management technique in modern computer architecture: Dynamic Voltage and Frequency Scaling (DVFS), where the voltage and the frequency of a microprocessor is adjusted on-the-fly, increased or decreased, depending upon circumstances. A decrease of the voltage or frequency results in a power saving and decreases the performance.

The authors of [29] propose a Q-learning based strategy applied to manage DVFS on a SoC in order to reduce the energy consumption. The idea is to adapt the frequency applied to the workers according to the output buffer filling. The model uses the output buffer filling b_t at given time step t as a state for the agent. The state space size is reduced by discretizing the level of the buffer filling in $N+1$ equal parts. The applied voltage has the minimum value that supports the frequency f_t, hence it is not part of the action space. The authors consider a set of $M+1$ frequencies between f_{min} typically a hardware constraint and f_{max}, leading to a set of actions $A = \{a_j\}_{j=0,M}$ where M is a hardware constraint.

The reward value is depending of the buffer filling and it is composed by two parts: one ranging from 0 to the set-point and the other one ranging from the set-point to the buffer size, as in Eq. 2.3 :

$$reward(b) = \begin{cases} p_1^2 b^2 + r_1 b + q_1 & b \in [0, \text{set-point}) \\ p_2^2 b^2 + r_1 b + q_2 & b \in [\text{set-point}, B) \end{cases} \tag{2.3}$$

The parameters $\{p, q, r\}_{1,2}$ are chosen empirically such that the reward is negative when the buffer filling is lower that 15% from the buffer size or higher that 95% from the maximum buffer size.

This approach was evaluated on a real test board with an ARM host processor and a SoC with 16 processing elements. The application is a part of HMAX, an object re-

cognition application. Two main metrics are used to illustrate the performance of the DVFS manager: the normalised energy consumed by the application including the energy overhead of the manager and the number of dropped tokens per second (indicating how many times the throughput of the application is not respected).

The performances of this approach are compared to the state of the art, a proportional integral (PI) controller [30] and a non-linear, threshold-based controller [31]. On this application, it outperforms the state-of-the-art in term of energy consumption. The energy consumed is 15% lower than with the PI controller and 44% compared to the non-linear controller. The dropped tokens is 30% lower than with the non-linear controller, but the PI controller outperforms since it is designed to minimize this performance specifically. The time overhead of the proposed controller varies from 0.6% to 1.2% depending on the state space sizes. Instead of modifying the performances, one solution is to change the duty cycling.

Nodes are tiny sensors which operate with limited power. Once they run out of energy, they become useless. If too many nodes are out of energy, the WSN cannot work properly. A solution can be to plan the sleep of nodes. Thus, active nodes operate normally and sleep nodes recharge their battery using harvesting devices. In [21], a reinforcement learning-based sleep scheduling for coverage (RLSSC) algorithm is proposed for sustainable time-slotted operation in rechargeable sensor networks. A part of the nodes enters into sleep mode to preserve the network lifetime, and the other ensures the desired area coverage.

Each node is an agent and chooses to enter into sleeping or active mode. The state space is composed of s_L, s_K, s_E which represent the state of the light condition, distance to energy recharging balancing and the current energy of the nodes, respectively. Each action is rewarded depending on the battery's charge:

$$r_{high} = a \cdot \left(\frac{2}{1 + \exp(-b(s_K + \xi)} - 1 \right) \tag{2.4}$$

$$r_{moderate} = a \cdot \left(\frac{2}{1 + \exp(-bs_K)} - 1 \right) \tag{2.5}$$

$$r_{low} = a \cdot (1 - s_L) \cdot \left(\frac{4}{(1 + \exp(-bs_K))(1 + \exp(bs_K))} - 1 \right) + \ldots \\ a \cdot s_L \cdot \left(\frac{2}{1 + \exp(-bs_K)} - 1 \right) \tag{2.6}$$

where r_{low}, $r_{moderate}$ and r_{high} represent the low, moderate and high-level of s_E, respectively. a and b are tuning parameters and ξ is the distance to the origin.

The proposed method is compared with LEACH algorithm [32] and random selection. Random selection is the algorithm where the active nodes are selected randomly from the group members. The results show that RLSSC can not only adapt to the dynamic environment but also can balance the energy between sensor nodes in real-time. Since the energy consumption is balanced between the node, the network lifetime increased by 20% as compared to LEACH. Moreover, the coverage of RLSSC is relatively stable compared with random and only about 1% less compared with LEACH.

Nodes usually operate in networks thus, a lot of work is being done to optimize the communications between nodes. Part of the work includes the optimization of the data routing within networks, which is discussed in what follows.

2.2.2 Routing protocols optimisation

In a WSN, the lifetime of the network is an important issue and the communication are the most energy consuming task of the nodes. There are different definitions for the lifetime, (1) the time until the first dead node appears; (2) the time until the first isolated node appears; and (3) the time until the network cannot accomplish any packet delivery. An isolated node has energy, but no path to the sink, all the neighbouring nodes have died. Many works used RL algorithms to adapt the packet routing in the network in order to prolong its lifetime and to increase the packet delivery.

One of the early approaches using a RL algorithm to solve the routing problem appears in [33] in the context of a wired network. The algorithm, presented as Q-routing, uses a distributed approach which gathers estimated delay information from immediate neighbours to make the routing decision. It learns a routing policy which balances minimizing the number of "hops" a packet will take with the possibility of congestion along popular routes. The final objective is to minimize the total delivery time. Simulation results show that under high network load, this algorithm outperforms the shortest-path algorithm ([22], [34]), it maintains the average delivery time under a network load level twice as high, and even performs well under changing network topology.

In Underwater Acoustic Networks (UANs), maximizing network lifetime is a key requirement. Accordingly, [35] propose a RL based approach that aims to distribute traffic among sensors to improve the lifetime of the network. In this work, the system state re-

lated to a packet is defined as the node that holds the packet. So s_i denotes the state of a packet held by node i. The action taken by node i to forward a packet to node j is denoted as a_j. If this action is successful, the transition from state s_i to state s_j with the probability of P_{ij}^j and stays in the same state s_i with the probability of $P_{ii}^j = 1 - P_{ij}^j$, if it fails. Though these transition probabilities are unknown, the authors argue that this can be estimated at run-time from the history. Accordingly, the overall reward function at time step t can be defined as follows,

$$r_t = P_{ij}^j R_{ij}^j + P_{ii}^j R_{ii}^j \tag{2.7}$$

where R_{ij}^j is the reward when the transmission is successful:

$$R_{ij}^j = -c - \alpha_1(E_i + E_j) + \alpha_2(D_i + D_j) \tag{2.8}$$

where α_1 and α_2 are tunable weights and c is the constant cost associated with consumption of resource (bandwidth, energy, ...) when a node chooses to transmit. E_i is the cost function associated with residual energy (E_i^{res}) and initial energy (E_i^{ini}). The energy's cost function penalizes the system when residual energy decreases and is defined as,

$$E_i = 1 - \frac{E_i^{res}}{E_i^{ini}} \tag{2.9}$$

Similarly, D_i is defined to measure the energy distribution balance as follows,

$$D_i = \frac{2}{\pi} \arctan(E_i^{res} - \bar{E}_i) \tag{2.10}$$

where $\bar{E}_i$ is the average residual energy of the node i and all its direct neighbours. This parameter increases the chance of neighbours with higher residual energy being preferred. The reward function for the case where a packet forwarding attempt fails is defined as,

$$R_{ii}^j = -c - \beta_1 E_i + \beta_2 D_i \tag{2.11}$$

where β_1 and β_2 are again tunable weights. The authors use Q-learning at each node to enable them to learn about the environment using control packets and take action to improve network lifetime. The proposed solution is shown to outperform the vector-based forwarding protocol [36], a geographical routing protocol designed for UANs by

achieving 20% longer lifetime. The authors claim the proposed solution can be applied for various UAN applications by tuning the trade-off between latency and energy efficiency for lifetime.

With a similar approach, [37] proposes a reinforcement-learning-based routing (RLBR) protocol to increase the number of packet delivery over energy consumption and extend the network lifetime according to the three aspects (1), (2) and (3) [1]. In this work, the system state related to a packet is defined as the node that holds the packet. So s_i denotes the state of a packet held by node i. The action taken by node i to forward a packet to node j is denoted as a_j.

The reward depends on the residual energy of the neighbour node E_j and the hop count from this neighbour node to the sink h_j. Both of these can be obtained from the neighbour table. The reward function is defined as follows:

$$R(i,j) = \frac{E_j}{d^n(i,j) \times h_j} \tag{2.12}$$

where $d(i,j)$ is the distance between the current node and this neighbour node and can be computed according to equation 2.13. In addition, n is a constant and its value is shown in Eq. 2.14.

$$d(i,j) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \tag{2.13}$$

where (x_i, y_i) and (x_j, y_j) are the location coordinates of the current node and the neighbour node respectively.

$$n = \begin{cases} 2 & d \leq d_0 \\ 4 & otherwise \end{cases} \tag{2.14}$$

where d_0 is a constant of distance threshold.

The proposed algorithm was compared to four different state-of-the-art algorithms, i.e. energy-aware routing (EAR) [38], balanced energy efficient routing (BEER) [39], Q-Routing [33], and multi-agent reinforcement learning-based self-configuration and self-optimization (MRL-SCSO) [40]. The simulation results show an improvement in the network lifetime according to the three aspects. On average, the RLBR protocol improves the time before the first dead node appears by 200%, 88%, 7% and 400% over EAR, BEER, MRL-SCSO, and Q-routing, respectively. The time before the first isolated node appears is increased by 289%, 84%, 13%, and 338% over EAR, BEER, MRL-SCSO, and

1. (1), (2), (3) are the definition of a lifetime presented at the beginning of subsection 2.2.2.

Q-routing, respectively. And the time until the network cannot delivery any packets is increased by 117%, 85%, 25% and 78% over EAR, BEER, MRL-SCSO, and Q-routing ,respectively; Table 2.1 summarises the results according the lifetime aspect (1) to (3).

TABLE 2.1 – Performance comparison of RLBR [37] and state-of-the-art according to the different definitions of a network lifetime

Aspect	(1)	(2)	(3)
EAR	200%	289%	117%
BEER	88%	84%	85%
MRL-SCSO	7%	13%	25%
Q-routing	400%	338%	78%

Moreover, this approach outperforms the state-of-the-art in terms of energy efficiency (i.e. the number of packets delivery per energy unit). At first, RLBR and MRL-SCSO deliver less packets than the other approaches due to initial learning. However, through continuous learning, they find the most appropriate path to transmit the packet and the difference of packet delivery between RLBR and the other protocols becomes more obvious. The performance of the proposed protocol is due to different factors. First, the reward is influenced by the distance between the current node and its neighbours. If the distance is greater than the threshold, the reward decreases more quickly. Thus, the probability of taking a close node as forwarder is higher. Consequently, the energy consumption for the current node to send a packet to the next forwarder is lower. Second, the protocol has a lower overhead since it does not need to build and maintain routing table. Finally, the scheme of data packet carrying feedback can further save energy. For the packet delivery, RLBR considers the hop count to the sink to define the reward function to encourage nodes to select the next forwarder nearer to the sink. Such a way quickens the packet delivery and decreases packet loss and ultimately achieves an increase of packet delivery.

In addition to the optimization of the packet routing in WSN, the RL approach has been also applied to the Media Access Control (MAC) layer to improve the energy efficiency of the communications, as discussed in what follows.

2.2.3 Communications optimisation

Since the communication task consumes a lot of energy, another communication based strategy tries to improve resources allocations, in particular at the MAC layer.

With the objective to increase the energy efficiency of communications, [16] proposes a new RL based protocol called RL-MAC. The objective of this protocol is to maximize an energy efficiency metric (i.e. the ratio of effective transmit/receive time to the total reserved active time) and to maximize the data throughput. Moreover, the protocol must avoid early sleeping phenomenon. An early sleep occurs in scenarii whereby a node go to sleep when a neighbour still has packets designated for it; as a result, the node will miss all packets designated for it.

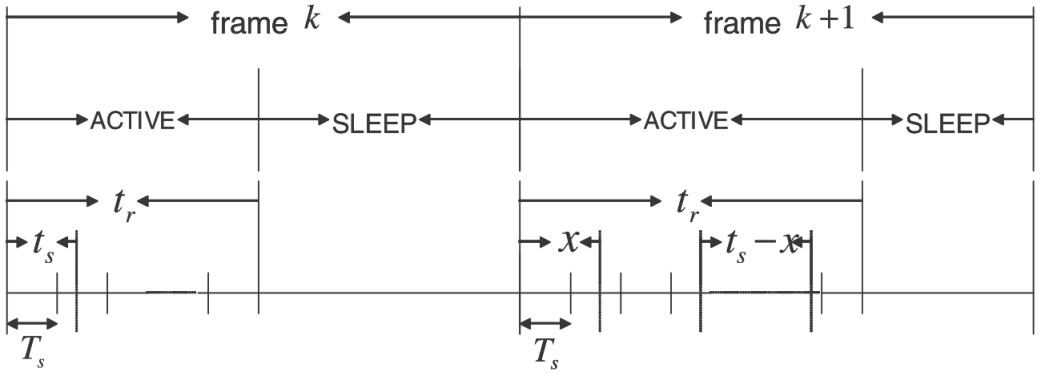

FIGURE 2.2 – Frame structure employed by the RL-MAC protocol [16]

This protocol employs a frame-based structure (Fig. 2.2). The frame is composed of two parts: when the node is active and when it sleeps. The frame is itself divided into finer time slot T_s. In active time, the node listens to the channel and attempts to exchange packets with its neighbours. At the beginning of each frame, the RL agent dynamically reserves slots as active time. The set of actions available for the node is the set of reserved active time, and the state space is the number of packets in the buffer.

The reward is designed as follows:

$$r_k(n_b, t_r) = \begin{cases} \frac{(n_s+n_r+1)\cdot T_p}{t_r-t_s} - \eta\frac{n'_b-n_b}{\sqrt{B}} & t_r, n_b \neq 0, n'_b > n_b \\ \frac{(n_s+n_r+1)\cdot T_p}{t_r-t_s} & t_r, n_b \neq 0, n'_b \leq n_b \\ -\eta\frac{n'_b-n_b}{\sqrt{B}} & t_r, n_b = 0, n'_b \neq n_b \\ 1 & t_r, n_b = 0, n'_b = 0 \end{cases} \tag{2.15}$$

where n_b is the number of packets in the buffer at the beginning of the frame and n'_b at the end, t_r is the active time reserved, B is the size of the buffer, T_p is the packet transmission time, n_s and n_r the number of packets sent and received during this frame, respectively, and η is a weight.

This protocol is compared to a state-of-the-art protocol, called S-MAC [41], according to three performance metrics: efficiency, throughput and latency. Simulations for different WSN topology (star and linear) have been conducted.

In both topology, RL-MAC offers, on average, a better energy efficiency compared to S-MAC. The energy saving is over 50%. In both algorithm, the energy efficiency increases when the traffic load increases. Moreover, RL-MAC can achieve a much higher throughput than S-MAC when the load is heavy. The throughput of RL-MAC is 357% and 246% higher than the throughput of S-MAC in star and linear topology, respectively. This is due to the fact that RL-MAC adaptively increases the reserve active time in response to increased traffic load. RL-MAC achieves a lower latency on average.

The communication channel adds noise when data is transmitted; if the transmission power is too low, the data cannot be received. Thus, [42] proposes an approach where the channel state is taken into account. An RL agent chooses the best operating mode (OM) (Table 2.2) and when the communication is possible, it also chooses the transmission power and the number of packets transmitted. It uses, as state space, a combination of the channel state $g_i \in \{1, 2, \ldots, K\}$ and the buffer state $b_i \in \{1, 2, \ldots, K\}$. The state of the node in time slot i is noted $s_i = g_i, b_i$.

$$a_i = \begin{cases} (A_{m_i}, P_{t,i}, c_i) & m_i \in \{0, 1\} \\ A_{m_i} & m_i \in \{2, 3, 4, 5, 6, 7\} \end{cases} \tag{2.16}$$

where A_{m_i} is the operating mode at time slot i, $P_{t,i}$ is the transmission power and c_i is the number of packets transmitted.

$$c_i \begin{cases} = 0 & \text{Defer} \\ \in 1, 2, 3, 4 & \text{Transmit} \\ \in Fr(01) \cup Fr(12) \cup Fr(23) \cup Fr(34) & \text{Fragment transmit} \end{cases} \tag{2.17}$$

The action of transmit means that the node sends only one to four complete packets. A full-sized packet can be broken into n equal-sized frames. Then, the action of fragment transmit means that the node will send zero to three complete packets plus 1 to $n - 1$ fragments.

If a non-fragment action a_i is taken in the system state s_i, the cost function is defined as:

$$R = \frac{P_i \times T_p}{U_i} + \beta_1 D(b_i, a_i) + \beta_2 S(A_{m_{i-1}}, A_{m_i}) + \varphi \tag{2.18}$$

OM \ Unit	Processor	Sensing unit	Radio
A_0	Active	On	tx, rx
A_1	Active	Off	tx, rx
A_2	Idle	On	rx
A_3	Idle	Off	rx
A_4	Sleep	On	rx
A_5	Sleep	Off	rx
A_6	Sleep	On	Off
A_7	Sleep	Off	Off

TABLE 2.2 – Different operating mode of the sensor node [42]

where

- P_i : power consumption
- T_p : transmitted packet
- U_i : utility (throughput)
- $D(b_i, a_i)$: denotes the average number of packets dropped in time slot i because of buffer overflow
- $S(A_{m_{i-1}}, A_{m_i})$: the expected energy consumption due to switching from OM $A_{m_{i-1}}$ to OM A_{m_i}
- φ : the fragment cost
- β_1 and β_2 : constant weight

For the fragment transmission scheme, a full-size packet is divided into n equally sized fragments. This scheme improves the energy efficiency and reduces the frame error rate. A state clustering approach is also used to reduce the size of the proposed MDP.

Three policies were compared: the first, no packet fragmentation possible; the second, packet fragmentation possible and the last, always-on policy, the node operates only in three modes (i.e. A_0, A_2 and A_4). The best performance is obtained with the always on policy which presents, logically, a higher throughput, 47.3% and 21.7% with a SNR to 1 as compared to the first and second policies, respectively. The difference is only 7.1% with the others policies when the SNR is 8. Moreover, the always on policy outperforms in packet loss rate, 22.1% and 13.2 when compared the policy without or with the packet fragmentation possible, respectively, with a SNR set to 1 and 10% with the SNR is 8. However, the policy with data fragmentation is the most energy efficient approach. It outperforms the always on policy by 48.6% and 12.5% with the SNR to 1 or 8, respectively. The difference with the policy without fragmentation is lower, 6.1% and 2.3%, with the

SNR set to 1 or 8.

Energy management and communication optimization are not always sufficient to extend a network lifetime. The addition of harvesting capabilities increases the energetic resources, but the energy harvested varies over time according to changes in the environment. Different algorithms try to predict the energy generation and some approaches rely on RL methods.

2.2.4 RL and energy harvesting

Another application of RL is found in energy harvesting management. Energy harvesting capabilities complement the battery to extend the system lifetime. Solar energy is the most effective environmental energy for harvesting because of its high energy density, nevertheless it comes from a non-controllable source, the sun. In this context, [43] presents a prediction algorithm (QL-SEP) of the energy generation from solar harvesters based on the Q-learning algorithm. Solar energy is a periodic energy source in which the time domain can be split into equal-length slots repeated daily. Exponentially-Weighted Moving Average (EWMA) [44] is the most used algorithm and has inspired the development of many prediction approaches. The EWMA considers the historical information of an energy generation profile combining the energy estimated and the energy harvested as presented in Equation 2.19:

$$E(d,n) = \alpha E(d-1,n) + (1-\alpha)H(d-1,n) \tag{2.19}$$

where d represents the current day and n is the slot number. $0 < \alpha < 1$ is a weighting factor which balances the importance of the estimated energy E and the last amount of harvested energy H. EWMA is an efficient way of observing long-term seasonal conditions with no mechanism for adapting to relatively short-term variations. The proposed approach [44] updates Equation 2.19 with a new parameter, called the daily ratio (DR):

$$E_{QL_SEP} = E_{EWMA} \cdot (1 + DR) \tag{2.20}$$

The DR represents the trend in the current solar energy generation, investigating the

behaviour of the solar energy in the recent slot.

$$DR = \frac{\sum_{i=1}^{N} \left(\frac{H-P}{P}\right) \cdot Q(i) \cdot i}{\sum i} \tag{2.21}$$

where P indicates the prediction energy from QL-SEP, H is the actual harvested energy in the slot and N is the number of time slot taken into account. The Q-value denotes the reliability of the prediction.

The Q-value of a slot is updated at the end of the slot in association with the Overall Prediction Error Ratio (OPER) in 24 slots. A Prediction Error Ratio (PER) in a slot is compared with OPER. If PER is lower than OPER, the reward is a positive value (+1), otherwise the reward takes a negative value (-1). The PER for a single slot is calculated as:

$$PER = \left|\frac{H-P}{P}\right| \tag{2.22}$$

The QL-SEP is compared to three state-of the art prediction algorithms: EWMA, Accurate Solar Energy Allocation (ASEA) [45] and Profile Energy Prediction Model (Pro-Energy) [46]. The author conducts a simulation over a year. The QL-SEP outperforms the others approaches with an average PER of only 0.27 (Table 2.3).

Approach	Average PER
QL-SEP	0.27
ASEA	0.36
EWMA	0.4
Pro-Energy	0.57

TABLE 2.3 – Performance comparison of QL-SEP with state-of-the-art

The RL approaches are able to adapt the energy consumption or predict the harvested energy in a dynamic environment using a lookup table. In this section, we have presented approaches where the reward is given after each action. In the following section, we present a different approach using eligibility trace to reward a sequence of actions.

2.3 RL algorithms with traces

When an agent performs an action with Q-learning, it receives an immediate reward r_t that evaluates the efficiency of the action a_t. However, an agent may receive a reward

only after performing a sequence of actions, assigning credit to the appropriate state-action pairs becomes an issue. To solve this, different algorithms introduce a memory variable for each state-action pair called the eligibility trace. During each epoch, the eligibility trace, for all state-action pairs, decays by $\gamma\lambda$ (where λ range between $0 < \lambda < 1$), is a parameter that allows specifying the strength with which Q-values of early state-action pairs are updated as a consequence of the final reward.

2.3.1 Energy management with RL based on traces

[47] uses the above approach to propose an on-line power management technique for peripheral devices. The idea is to adapt the energy consumption according to the workload with no prior information. Indeed, devices have different operating behaviours and performance evaluation. Their technique adjusts on the fly the energy consumption and takes into account uncertainties that emanate from hardware and application characteristics. Moreover, the authors add a workload prediction based on on-line Bayes network to improve the performance of their algorithm. In order to maximize a node's lifetime, the node is equipped with one or more energy harvesting devices, enabling the nodes to be entirely powered by the energy harvested in their environment.

Simulations have been done for two different devices: Hard Drive Disk (HDD) and a wireless adapter card (WLAN). For the HDD, the proposed approach can achieve much lower power consumption than the references, the maximum power saving with the same average latency is 18.1%. For the WLAN card, several traces have been used for on-line video watching, web surfing, on-line chatting and a combination of web surfing, on-line chatting and server accessing. The correct prediction rate of the on-line Bayes predictor can be 99.2% for the video trace, 79.8% for the web trace, 82.8% for the combined trace. In comparison, the correct prediction rate of an exponential predictor [48] for the combined trace is less than 65%. Moreover, the maximum power saving with the same latency is 16.7%; while the maximum latency saving with the same power consumption is 28.6%.

In [15], the author uses an other RL algorithm, SARSA(λ) [49], to achieve an Energy Neutral Operation (ENO) power management of a sensor node in a monitoring application. ENO is a mode of operation where the energy consumption of the node is always at most the energy than has been harvested from the environment. In order to achieve energy neutral operation, energy optimisation methods need to fulfill the energy neutrality constraints while maximising performance.

The node adapts dynamically its power consumption depending on the energy har-

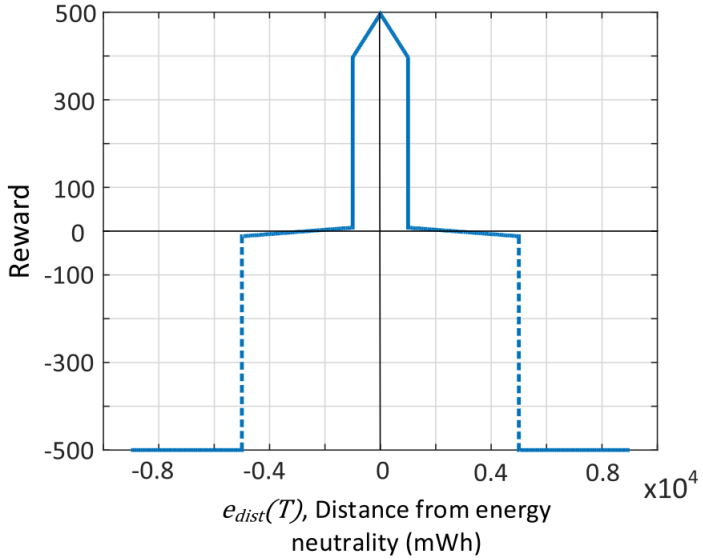

FIGURE 2.3 – Reward function [15]

vesting opportunities by varying its duty cycle to ensure ENO. Each day the node receives weather forecast information. Its set of action, A, is defined as the set of discrete duty cycles that can be chosen. $A \in [D_{min}, D_{max}]$, where D_{min} and D_{max} are the minimum and maximum duty cycle of the sensor node. In this work, the authors use $A = \{20\%, 40\%, 60\%, 80\%, 100\%\}$. The different states in which the agent can exist is given by combination of the battery charge S_{batt}, the distance to ENO S_{dist}, the energy harvested $S_{eharvest}$ and the weather the agent may expect S_{day} : $(S_{batt}, S_{dist}, S_{eharvest}, S_{day}) \in S$. The reward is depending of the distance between the battery charge and the ENO at the end of the day (Fig. 2.3).

The authors compare the proposed policy to a power management strategy [44] referred as Offline policy. Offline policy uses linear programming optimization methods with non-causal data on energy harvesting opportunities to determine the optimal duty cycles. The compared policy uses an optimization window of one day (24 hours) to calculate the duty cycles, and their SARSA(λ) agent is also trained in one-day period. The proposed method achieves less than 6% root mean square deviation from ENO. Offline policy achieves 3.46% deviation from ENO and with a Naïve policy (Battery-Centric), more than 23% deviation occurs. The use of SARSA(λ) results in a highly adaptive behaviour. The node is able to adapt its energy consumption to the seasonal variation, climatic changes and more important, changes in device performances (i.e. degradation in the node's energy efficiency or in harvesting capabilities) or battery degradation.

Another domain of application is task scheduling. Thus, [50] uses an algorithm with eligibility traces in an energy aware task scheduling for a target tracking application. In such an application, the node needs to perform different actions such as sensing, sleeping, communicating, etc. Each task has an impact on the overall performance of the application, and when the performance increases, the energy consumption increases as well. In this work, an agent chooses an action with the objective to balance these different aspects.

The set of action available is composed of six actions: target detection, target tracking, communication with the neighbour, trajectory prediction, trajectory intersects and sleeping. The environment is composed of three different states. The idle state occurs when there is no currently detected target and no object is detected by any neighbour. When there is no currently detected targets, this is the state of awareness. The sensor nodes can still receive some trajectory information by which they can decide that the expected arrival time of at least one target. When there is at least one detected target within the Field Of View (FOV) of the node, this is the tracking state.

The agent is rewarded with the level of battery charge and the number of tracked target, as defined in Eq. 2.23 :

$$r = \beta \left(\frac{E_i}{E_{max}} \right) + (1 - \beta) \left(\frac{P_t}{P} \right) \tag{2.23}$$

where the parameter β balances the conflicting objectives between E_i and P_t. E_i represents the residual energy of the node. P_t represents the number of tracked position of the target inside the FOV of the node. E_{max} is the maximum energy level of sensor node and P is the number of all possible detected target's positions in the FOV.

In that work, a variant of TD(λ), named True Online TD(λ) (TOTD(λ)), is used to determine the best policy. This algorithm is compared to state-of-the-art algorithm: a distributed independent reinforcement learning named DIRL[51], a cooperative reinforcement learning named cooperative Q-learning [52], a cooperative reinforcement learning SARSA(λ) [53] and an adversarial algorithm Exp3 (Exponential-weight algorithm for exploration and exploitation) [54]. Four different simulations were conducted to find out the trade-off between tracking quality and energy consumption with different network size, different randomness of moving targets and an evaluation of the average execution time and average communication effort.

In term of tracking quality, the proposed approached outperforms all the methods, while achieving state-of-the-art performances in term of energy consumption. When the

network size increases, the tracking quality increases and the energy consumption too. However, the proposed approach offers the best trade-off between tracking quality and energy consumption. When the randomness of the moving targets increases, the TOTD(λ) outperforms. The DIRL and cooperative Q-learning are resources-aware in term of execution time and communication effort. Exp3 and TOTD(λ) requires 25% and SARSA(λ) requires 86% longer execution times, respectively.

The algorithm using eligibility traces is not very common in the state-of-the-art of energy management. This is due to the need in computation capacity. The convergence of the Q-values is very slow. Indeed, most of the eligibility traces are close to zero. Moreover, such approaches require to know all information from every time step of the agent's sequence of actions and states. Nevertheless, these algorithms are very useful when the reward is not available immediately at the next time step.

Another popular approach in RL uses Neural Networks to compute the Q-value of the different action instead of maintaining a table, as presented in what follows.

2.4 Neural network approaches in RL

When the environment becomes too large, the use of a look-up table becomes inefficient. Instead of storing the values, a more efficient approach is to use neural networks to approximate the policy function and find the best action to choose. Neural networks became widely popular with the success of Deep Q-learning on Atari games [8], and even more since this approach was also used in AlphaGo [55], which has succeeded in beating the human in the game of Go. And in this way, they have been widely used in energy optimisation.

2.4.1 Energy management using neural network approaches in RL

Neural Networks have been applied in task scheduling, particularly. In this way, [56] proposes an energy-efficient scheduling scheme based on deep Q-learning for periodic multi-tasks in real-time systems (DQL-EES). The proposed method combines a stacked auto-encoder and a Deep Q-learning model. The stacked auto-encoder is an unsupervised learning technique which is used in that work to learn the features of each input system state. Then, the agent uses the Deep Q-learning model to select the most appropriate

frequency and voltage. The different state is composed of the system utilization workload and the dynamic slack. The stacked auto-encoder $SAE(\Theta)$ is used to learn the features of each input system state and the Q-learning aims to compute the value of each action given the input system state. A memory replay is added to the Deep Q-learning model to avoid forgetting previous experiences.

The penalty is defined as:

$$p_t(s_t, a_t) = \frac{E_n(s_t, a_t)}{E_{t_{hyp}}} \tag{2.24}$$

where $E_n(s_t, a_t)$ denotes the total power consumption at the system state s_t using the DVFS technique. $E_{t_{hyp}}$ denotes the sum of the actual execution time of every task in a hyperperiod (i.e. the time between 2 scheduling decision).

The performance of the proposed algorithm is evaluated by comparing with a Q-learning-based hybrid scheduling algorithm (QL-HDS) [57] on different simulation task sets. Results demonstrated that the proposed algorithm can save, on average, 4.2% of energy compared to QL-HDS. However, the execution time is about 12.5% higher on average as compared to QL-HDS.

Nodes have a limited computing, memory and energy resources and some applications need a low latency and large bandwidth. To overcome these issues, a proposition is to process data as close as possible from the nodes to minimize the transmission, reduce the latency and optimize the bandwidth utilization. Nevertheless, a problem is to select an edge device to perform the computation from all the potential devices candidates within the radio coverage area. Thus, [58] presents a RL-based computation offloading scheme.

The agent selects the proportion of the data offloaded and the edge device which will performs the computation. The decision is depending on the radio transmission rate, the harvested energy and the current battery level. The utility of an action is evaluated using Eq. 2.25 :

$$U_i^{(k)}(x) = xC^{(k)} - \psi I\left(b^{(k+1)} = 0\right) - \beta E^{(k)} - \mu \mathcal{T}^{(k)} \tag{2.25}$$

where x is the proportion of the data offloaded to the edge device, ψ is the task drop loss, b is the battery level, β and μ are weighting parameters, $E^{(k)}$ is the energy consumption, $\mathcal{T}^{(k)}$ is the computation delay and $I(\varpi)$ is the indicator function which equals 1 if ϖ is true and 0 otherwise.

That work also uses the Deep Q-learning algorithm [8]. Two optimizations are proposed to improve the performance of this method. First, several convolutional layers compress

the state space and accelerate the convergence speed. Second, the authors use a transfer learning method to accelerate the learning speed; the Q-values are initialized with the offloading experience in similar environments.

The approach is compared to state-of-the-art algorithms a Q-learning scheme [59] and DRL [60], but also to a non-offloading scheme. The method proposed in [58] achieves the optimal computation offloading performance after convergence. Moreover, it outperforms the other approaches in term of energy consumption, computation delay and task drop rate. A version without NN is compared to the Q-learning. It reduces the energy consumption by 28%, the computation delay by 16.7%, and the task drop rate by 25%. The computation offloading with Deep learning approach further improves the performance, e.g., it reduces the energy consumption by 75.0%, computation delay by 32.6%, and the task drop rate of the IoT device by 85.6%, compared to the previous scheme. And when compared to DRL, it reduces the energy consumption by 58.3%, the computation delay by 26.7%, the task drop rate of the IoT device by 55.5%. The proposed approach improves the computation performance by incorporating more system state and the energy harvesting technique. Moreover, the method presented uses 345 MB of memory, the policy selection takes 8.3 ms and it converges in 400 time slot.

NNs are also used in the communication optimisation, to reduce the radio consumption or for routing packet in a network, which is discussed in what follows.

2.4.2 Communication optimisation

NNs are also used to minimize the energy consumed during the communication. In this way, [19] presents a method to reduce the energy consumption of the front-end radio using NNs. The proposed method consists in a real-time channel-adaptive system which is able to change its power consumption according to the channel state to achieve the desired level of Quality of Service. The key objective of learning based adaptation is to determine the optimum tuning knob combinations for the front-end for every channel's state on-the-fly.

First, an exploration determines the power consumed and the Error Vector Magnitude (EVM) for a channel state with particular tuning. Then, during a map phase, done on a general purpose processor, two neural networks are trained. The first one maps the power consumption with the tuning and the second one maps the EVM with the channel state and the tuning. The best tuning which respects the desired QoS and minimized the power

consumption are memorized in a on-chip LUT. Several iterations are needed, but at the end of the exploration, a tuning which minimizes the power consumption for a particular channel state is found. The exploration phase leads to additional power consumption. Nevertheless, the experiments show up to 2.5 times saving in energy consumption after the learning as compared to a worst-case design method.

NNs have also been apply on the routing problem in underwater acoustic sensor network (UASN), [23] proposes a protocol based on the Deep Q-learning algorithm, called Deep Q-Network-Based Energy and Latency-Aware Routing (DQELR), to take routing decision for packets. In an UASN, source nodes are deployed underwater to send collected data packets to sink nodes on the surface through relay nodes. Each node in the network comprises an agent; the current information of the sensor node, such as its residual energy, depth and neighbouring nodes, comprises the current state; and the forwarding of a packet from one node to the next node in the current state comprises an action. After the node sends the packet to one of its neighbours, it receives a reward, with the current state of each node updating to a new state. The reward is maximum if the packet is send to the sink.

The reward function can be defined as :

$$R(s_i, a_i) = \begin{cases} 100 & \text{if the packet is transmitted to the sink} \\ c + \alpha r_{sen} + \beta r_{dep} & \text{otherwise} \end{cases} \tag{2.26}$$

where α and β are parameters of the residual energy and depth, respectively. $c = \alpha + \beta$ which is much less than 100.

$$r_{sen} = \frac{e_{own}}{e_{max}} \tag{2.27}$$

$$r_{dep} = \frac{d_{own}}{d_{max}} \tag{2.28}$$

where e_{own} and d_{own} are the residual energy and depth of a neighbour node, respectively, and e_{max} and d_{max} are the maximum residual energy and maximum depth of all the neighbours, respectively.

The proposed method was evaluated in a simulation and compared to a Q-learning-based adaptive routing (QELAR) protocol [35] and a vector-based forwarding (VBF) routing protocol [61]. Results show that when the packet generation rate increases, the packet delivery ratio under each scheme decreases due to the congestion in the network and the packet collision rate. However, the DQELR achieves a packet delivery ratio and

an end-to-end latency similar to that of the state-of-the-art. The DQELR outperforms the other approaches with the energy efficiency and network lifetime. The DQELR improves the network lifetime by approximately $34 - 36\%$ as compared to the QELAR.

The optimization of the consumed energy is even more important in Wireless Body Area Network (WBAN) where the nodes are small and light to be worn. Thus, [62] proposes a deep reinforcement learning-based sensor access control algorithm (DRSAC) based on convolutionnal networks. The agent observes the state of the environment composed of the system's total Signal-to-Interference plus Noise Ratio (SINR) vector, denoted by τ; the transmission priority vector ρ; the battery level vector e; and the transmission delay vector g. Then, it chooses the transmission power of the selected sensor node.

Its action is rewarded using an utility function (2.29).

$$U = \sum_{i=1}^{M} \rho_i \tau_i - \alpha x_i \frac{l_i}{v_i} - \beta g_i \tag{2.29}$$

where M is the maximum number of sensors, v is the data transmission speed, l is the length of the data transmitted and x_i is the transmission power of the ith sensor. The coefficients α and β, where $\alpha, \beta > 0$, are used to adjust the influence of the energy consumption and the transmission delay.

The DRSAC is compared to LSE-TPC [63] and Q-learning approach [64]. The simulation shows that the proposed method outperforms. The Bit Error Rate (BER) of the DRSAC-based strategy is approximately 40.9% lower than the Q-learning-based strategy, and being approximately 53.6% lower than the LSE-TPC-based strategy. The energy consumption is 29.6% lower than the Q-learning-based strategy after their respective convergence, and approximately 34.5% lower than the LSE-TPC-based strategy. This method shows that the RL-based algorithm achieves an increased in performance while decreasing the energy consumption, i.e. the system becomes more energy efficient.

Look-up tables and NNs are not the only approaches used in RL; there are less common approaches to approximate the policy function in the literature, some of them are introduced in what follows.

2.5 Others approaches used in RL

Among all the approaches, a promising one is the use of temporal difference learning with linear function approximation in [65]. In that work, an energy manager called RLMan is proposed. RLMan dynamically adapts energy management policy to time-varying environment. The proposed approach is based on the temporal difference algorithm with an actor-critic structure. The author successfully applied it to the Pow Wow platform [66], a wireless sensor node with harvesting capabilities. The algorithm has been tested with two different sources of energy, indoor light and outdoor wind. RLMan has been compared to three state-of-the-art energy manager schemes that aim to maximize the throughput: P-FREEN [67], Fuzzyman[68] and LQ-Tracker [69]. P-FREEN and Fuzzyman require the tracking of the harvested energy in addition to the residual energy, and were therefore executed with perfect knowledge of this value. RLMan and LQ-Tracker were only fed with the value of the residual energy. In indoor and outdoor condition, RLMan and LQ-tracker achieve 99.9% of energy efficiency. When the energy buffer is reduced, RLMan outperforms LQ-tracker in term of throughput. The average throughput is more than 20% higher compared to LQ-Tracker in the case of indoor light, and almost 70% higher in the case of outdoor wind.

In [70], a bandit solver Exp3 [54] is used for the adaptive power allocation in device-to-device (D2D) communication. The action of each agent consists of a set of transmitting power levels. In this work, only three power levels are considered (low, medium and high). The state of D2D user u on resource block r at time t is defined as:

$$S_t^{u,r} = \gamma_r^c \cup G_{Bu} \cup G_{uv} \tag{2.30}$$

γ_r^c is the Signal to Interference plus Noise power Ratio (SINR) of a cellular user on the rth resource block. G_{Bu} is the channel gain between the base station and a user u. G_{uv} is the channel gain between two users u and v. The reward takes into account the system average channel capacity (Eq. 2.31).

$$R = \begin{cases} \frac{\sum_{u=1}^{U} log_2(1+SINR(u))}{U} & \text{if } \gamma_r^c \leq \tau_0,\ G_{Bu} \leq \tau_1 \text{ and } G_{uv} \leq \tau_2 \\ -1 & \text{else} \end{cases} \tag{2.31}$$

where U denotes the number of the users in the cell, $SINR(u)$ denotes the signal to interference plus noise power ratio of user u.

A simulation compares the proposed method with a random allocation and a distributed reinforcement learning. The proposed approach outperforms the two others; in particular the D2D throughput increased by 28% as compared with the distributed reinforcement learning.

2.6 Performance improvement in embedded systems thanks to RL

To summarize, RL has been shown to achieve various performance enhancements:

- Higher throughput. Higher throughput indicates higher packet delivery rate, higher successful packet transmission rate, lower packet loss rate and lower number of packet re-transmissions.
- Lower end-to-end delay/packet latency. Lower end-to-end delay and packet latency in single-hop and multi-hop transmissions, respectively, indicate lower number of packets in the buffer queue.
- Lower energy consumption. Lower energy consumption increases network lifetime. Since each sensor node operates on battery power, energy consumption is a common performance metric. Other performance enhancements, such as higher throughput and lower end-to-end delay, may indicate lower energy consumption due to lower packet loss rate and number of packet re-transmissions.
- Higher route discovery rate. Higher route discovery rate indicates higher success rate of finding a favourable route from a source node to a sink node.
- Higher in-contact time. Higher in-contact time indicates greater possibility of a sensor node to discover the presence of a mobile data collector node, as well as longer duration for data transmission, in a sleep-wake scheduling scheme.

Conclusion

The RL approaches provide a powerful tool for optimization in general, and more precisely for the energy consumption of embedded systems and communication between sensor nodes. It allows systems to adapt effectively their behaviour to changes in the environment.

Nevertheless, the development of such approaches faces many challenges. The conver-

gence rate, the minimization of learning cost (learning speed, how much did we waste before becoming efficient), the scalability of such methods are the main limitations of their use. Furthermore, the lack of guidelines for designers, non expert in machine learning, restricts the scope of RL in energy management in spite of the adaptability to a dynamic environment.

The choice of an algorithm is not trivial and designers need metrics to find the most suitable approach depending on their application and system. The different approaches are not appropriate for all applications and a given embedded system, but depend on memory and processing capacities. The choice of the appropriate approach for a designer is thus difficult, since there is a lack of metrics to compare the different solutions.

Moreover, the RL algorithms have a lot of parameters to configure: the learning rate, the discount factor, the initialisation of the learning, the reward function. The configuration is often done empirically using the experience of the designer; and few explanations are given in the literature making hard for a non-expert to understand the choice. The reward function defines the behaviour of the agent, it plays an important role in performance of RL approach. Nevertheless, the method to design the reward function is often not presented in the literature.

In this manuscript, in particular Chapters 3, 4 and 5, we present the exploration on the field of RL done during the thesis work. In the domain of embedded systems, there are constraints imposed by the system and the application. We try to provide designers guidelines to select the appropriate algorithm using different metrics. The conception of the reward function is not obvious and we compare different ones to find the best way to define a correct reward function. Moreover, we propose a reward function that adapts the reward depending on the energy available in the battery in order to balance performance and energy consumption. Finally, we explore a new approach for energy management using multi-agent approach for a single system.

In the following chapter, we present a comparison of different approaches using immediate reward to provides the guidelines to the designers for the choice of the approach to design an energy management algorithm for an embedded systems.

Chapter 3

Energy Management with Reinforcement Learning

Contents

3.1 Monitoring marine environment: a case study **54**

3.2 Presentation of the selected Markov Decision Process **55**

3.3 Look-up table approach with Q-learning **57**

3.4 Dyna Q-learning **59**

3.5 Neural network approach: Deep Q-learning **61**

3.6 Comparative results **64**

As seen in the previous chapter, there are two main RL approaches for the energy management: the first one uses a lot of memory, storing information in a look-up table, and the other one requires a lot of processing, training a network with the information. The choice of the approach to select when designing an energy management for an application is not obvious.

In this chapter, we address the question of the approach selection for a given but classical real case. We compare different versions of the popular Q-learning for the use case of a marine buoy equipped with solar panels to complement its battery. The versions chosen exploit either the look-up table approach or the neural network approach.

The case study, a marine buoy for environmental monitoring, is presented in the first section. The three following sections present the Q-learning algorithm and some variants, i.e., the Dyna Q-learning and the Deep Q-learning. Each section presents the algorithm and its performance for our use case. Finally, a comparative study is presented.

3.1 Monitoring marine environment: a case study

The marine environment is complex, and many practical applications need data from the sea such as wave height, water temperature, atmospheric pressure, wind speed or the presence of pollutants in water. The most reliable data source is the marine buoy measurements [71]. Nevertheless, marine buoys are difficult to access once deployed, that is why one has to be very careful with its energy resources to avoid failure. A solar panel is often used to complement the battery, but the energy harvested varies a lot (Fig. 3.1) and, therefore, an adaptive energy management is needed.

In this thesis, we consider a marine buoy (Fig. 3.2) deployed near to the coast communicating with a base station. The buoy is equipped with two sensors (Table 3.1), an 3D anemometer and an atmospheric sensor, to monitor environmental conditions. The sensors have different energetic behaviours and the buoy should be deployed for as long as possible. To complement the battery, the buoy is equipped with two small solar panels. To avoid collision with ships, a beacon light flashes for 500 msevery four seconds when the brightness is low.

Our aim is both to extend the buoy's lifetime and to maximize the number of measurements done by the sensors. To do so, the buoy should adjust the number of measurements of its sensors in order to preserve the battery's energy. Moreover, instead of modelling the energy consumption of the node in a laboratory, we exploit a RL method to adjust on

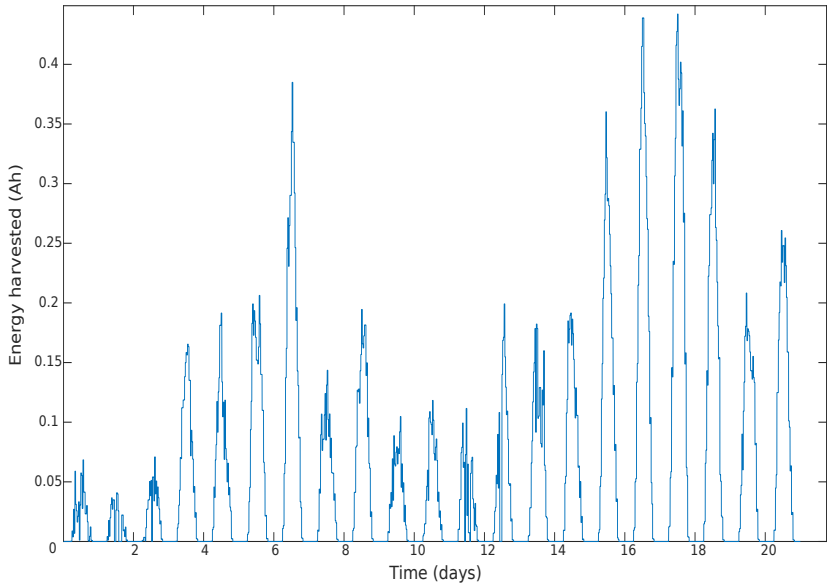

FIGURE 3.1 – Energy harvested in Lorient, France, by the buoy during a deployment of 21 days

the fly the frequency of the measurements. The node will adapt its behaviour according to the energy available and to changes in the environment. The choice of the algorithm is not easy since numerous algorithms exist. The Q-learning algorithm is popular and converges to optimality under conditions and, furthermore, different versions have been proposed over the years. The following sections present three versions using different approaches. Finally, a comparison is performed to find the most suitable version for the given application.

3.2 Presentation of the selected Markov Decision Process

For a fair comparison of the different algorithms, the same MDP was used. We define the selected MDP as follows: a state space $\mathcal{S}$ of 10 states (for each 10% increment of the battery's charge level), and a set of actions $\mathcal{A}$ for the different operating modes that the sensors are allowed to choose (i.e. a measurement frequency (Hz) $\in [0.1, 0.2, \cdots, 0.9, 1]$). The reward function indicates what kind of behaviour best serves our objective. In the proposed RL model, the reward awarded at the end of an episode depends on both the

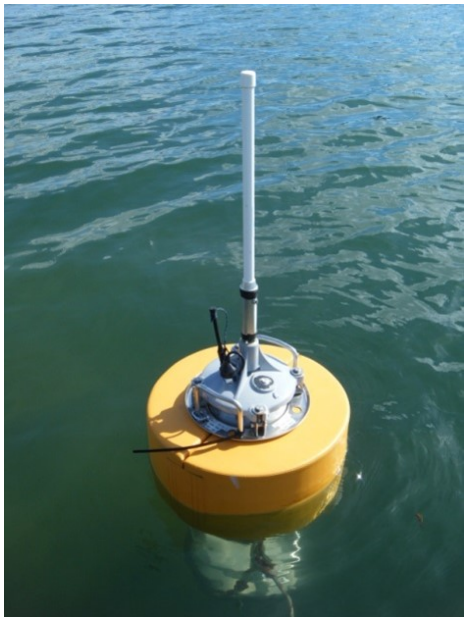

FIGURE 3.2 – Marine buoy

TABLE 3.1 – Buoy components

Components	**Device**
3D Anemometer	WindMaster HS
Atmospheric sensor	YOUNG61302L
Processor	Cortex-M4 MCU
Radio transceiver	CC1000
Energy harvester	**Power**
Solar panels	2×10W
Battery capacity	5200 mA/h

residual battery energy and the measurement frequency of the sensors during the episode. We reward the system using a simple function:

$$\eta \times \mathcal{N}(\text{frequency}) + (1 - \eta) \times \mathcal{N}(\text{battery charge level}) \tag{3.1}$$

where $\mathcal{N}$ is the function that normalizes the values and $\eta \in [0, 1]$ is a parameter which balances the importance given to the battery charge level and the sampling frequency. It is really important for the reward function to be deterministic and bounded, otherwise the algorithms may never converge.

3.3 Look-up table approach with Q-learning

The first evaluated algorithm is the Q-learning (Algorithm 1) [27]. At every time-step t, when the agent is in state s_t, it selects and performs action a_t that is derived from its policy, given its estimation of $Q(s_t, a_t)$. Once the agent has transitioned to the next state s_{t+1}, it receives the reward r_t for its action, and continues to estimate the Q-value, $\max_{a_{t+1} \in \mathcal{A}} Q(s_{t+1}, a_{t+1})$, in the next state. From there, the agent can compute the difference between its initial expectation of $Q(s_t, a_t)$ and the new estimation of the Q-value using:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right) \tag{3.2}$$

In Equation 3.2, the term $r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$ corresponds to the difference between the new and the old estimation of $Q(s_t, a)$, this is the *temporal-difference error*.

Algorithm 1 Q-learning [27]

```
Initialize Q(s, a) arbitrarily
The agent observes the initial state s_0
for each decision epochs do
    Choose a_t from s_t using policy π derived from Q
    Take action a_t, observe the new state s_{t+1} and the associated reward r_t
    Q(s_t, a_t) ← Q(s_t, a_t) + α (r_t + γ max_{a_{t+1}∈A} Q(s_{t+1}, a_{t+1}) − Q(s_t, a_t)))
    s_t ← s_{t+1}
end for
```

Learning rate α : The learning rate α determines how fast the new information will surpass the old one. A rate of 0 would not teach anything to the agent, whereas a rate of 1 would only teach the agent with the latest information. In our work, we decrease slowly the learning rate α in such a way that it reflects the degree to which a state-action pair has been chosen in the recent past. It is calculated as:

$$\alpha = \frac{\zeta}{visited(s, a)} \tag{3.3}$$

where ζ is a positive constant and $visited(s, a)$ represents the visited state-action pairs so far [72].

The value of α have an influence on the learning performance. Nevertheless, there is no method to easily find the best value for a given application. Moreover, depending on

the algorithm chosen, the influence of this parameter changes. Currently, α or γ and the others parameters are determined empirically. This is one drawback of this algorithm.

Simulation results for the look-up table approach with Q-learning

A simulation is conducted for a 21 days deployment of our buoy near Lorient, Bretagne, France. Figure 3.3 shows the evolution of the battery charge level and the sampling frequency of sensors with an iteration step of half an hour. This iteration step has been chosen to let the battery discharge between two decisions.

At the beginning (1), the agent has no prior knowledge of its environment and takes random actions to become award of it. After a few iterations (2), a daily variation is observed in the sampling frequency, it corresponds to the evolution of the harvested energy during the day. At the end of the simulation (4), the agent still do not know perfectly its environment but the daily variation are more well-defined. Around day 12 (3), the battery decreases and the agent still chooses the same action. This is due to fact that the agent stop to only explore its environment and it starts to exploit its knowledge; the learning rate has decreased. However, the last time it arrives in these states in day (day 2 to day 6), it learns to select these actions. It will need to more exploration of these states to takes better actions.

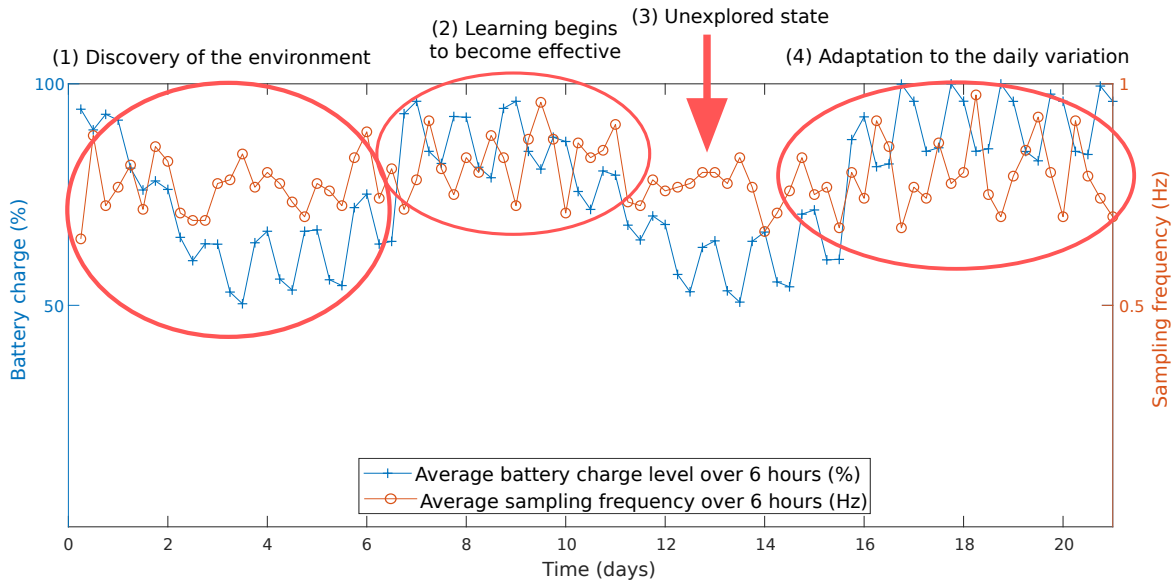

FIGURE 3.3 – Evolution of the battery charge level and sampling frequency of the sensors using the Q-learning algorithm

The agent succeeds in adapting itself to the daily variation of the harvested energy without prior knowledge about it. However, one problem with this algorithm is that we have to store the Q-value of each pair (s, a), and it can only be used for discrete state space and discrete action space. In addition, its convergence time is difficult to estimate and makes it unusable for applications with time constraints. The battery may be completely depleted before the learning process is completed. To improve the convergence's time of the Q-learning algorithm, different usable solutions have been proposed in the literature. In the following section, we present the Dyna Q-learning algorithm and the results of a simulation with identical parameters as Q-learning. The backbone of Dyna Q-learning algorithm is the Q-learning to which few adjustments are made. And the one advantage is the adjustment of its learning process, which makes it possible to adapt the learning speed according to the available processing power.

3.4 Dyna Q-learning

The Dyna Q-learning (Algorithm 2) [73] is a variant of the Q-learning, which also uses a look-up table to store the Q-values. One drawback of the Q-learning algorithm is its slow convergence; the Dyna Q-learning is proposed to accelerate this convergence. To achieve this, Dyna Q-Learning uses a partial and deterministic model of the environment to learn faster using the previous experiences. The last transition and the associated reward are stored in memory for each visited state. The algorithm uses this model to improve the evaluation function at each episode or even independently during a break in the decision making process. It selects a state s already visited and chooses an action a already performed, and then it uses the transition s_{t+1} and the reward r stored in the memory to update the Q-value $Q(s, a)$. The number of updates per sampling period is noted N.

We use the same parameters α, γ and time slot as for the Q-learning algorithm. Increasing the value of N reduces the learning phase up to a certain limit. However, this increases the number of computations as well, which results in an increase of the energy consumption. We choose to set N to 10, this value was determined after a series of experiments to improve the convergence rate; higher values did not accelerate the learning as much.

Algorithm 2 Dyna Q-learning [73]

Initialize $Q(s, a)$ arbitrarily
The agent observes the initial state s_0
for each decision epochs **do**
  Choose a from s using policy derived from Q
  Take action a, observe the new state s_{t+1} and the associated reward r
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right)$
  $m(s_t, a_t) \leftarrow s_{t+1}, r_t$
  for $i = 1$ **to** N **do**
    $s \leftarrow$ random visited state
    $a \leftarrow$ random visited action
    $s_{t+1}, r_t \leftarrow m(s_t, a_t)$
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right)$
  end for
end for

Simulation results for Dyna Q-learning

We simulate the Dyna Q-learning in the same conditions as for the Q-learning during 21 days. Figure 3.4 shows the evolution of the battery charge level and the sampling frequency of the sensors. At the beginning of the simulation (1), the agent has no prior information about its environment and takes random actions as can be seen with high variation in the sampling frequency. Around day 13, the sampling frequency follows the battery behaviour during the charge and discharge (2). At day 18, the sampling frequency reached the maximum at 1 Hz while the battery is fully charged (3). We can consider that the value converges around day 13.

The results show that the Dyna Q-learning algorithm increases the convergence speed of the Q-value by 17% in this example as compared to the Q-learning simulation. The agent can take better decision about its sampling frequency to adapt itself in function of the evolution of the battery load. However, as the convergence of the Q-values is accelerated, the memory requirement increases. Indeed we need a look-up table ($\mathcal{A} \times \mathcal{S}$) for the Q-values and 2 others of the same size to store the transitions and the associated rewards. The problem is that embedded systems have often small memory capacity. To reduce the use of memory, another approach consists in computing the different possible Q-values using a neural network such as Deep Q-learning. The memory only stores the weight of the neurons, allowing a larger MDP.

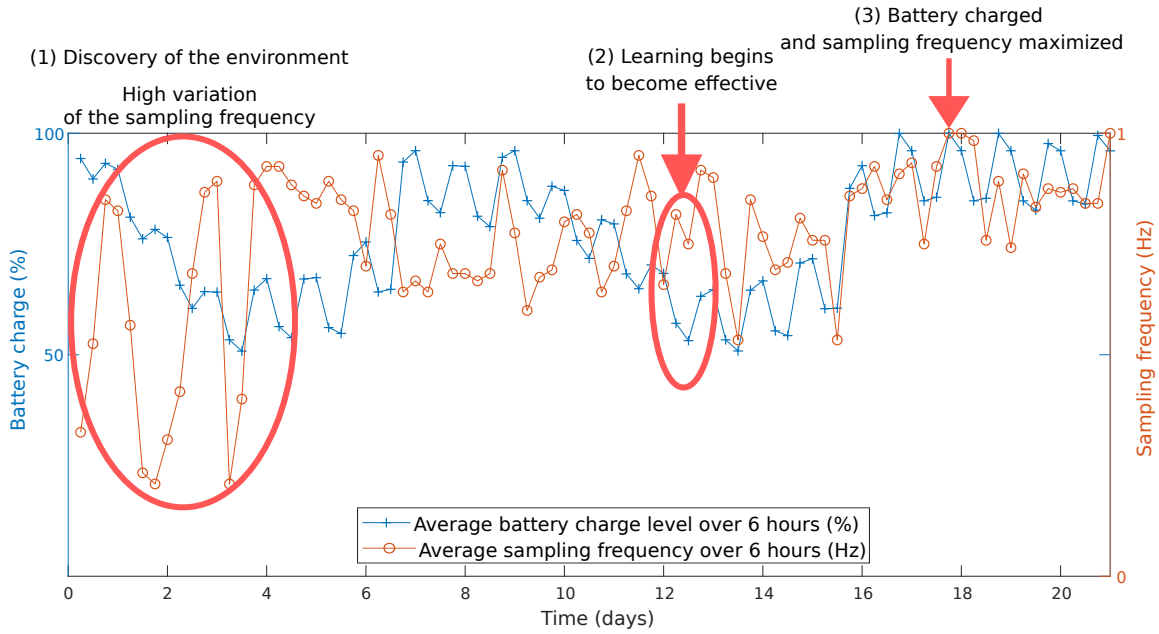

FIGURE 3.4 – Evolution of the battery charge and sampling frequency of the sensors using the Dyna Q-learning algorithm

3.5 Neural network approach: Deep Q-learning

Deep Q-learning (Fig. 3.5) [8] uses a neural network to take advantage of their ability to generalize the learning. Instead of storing the Q-value for each state-action pair in a look-up table, a neural network takes as an input the state and for each possible action computes the expected reward (Fig. 3.5). We take the biggest Q-value of this output to find our best action. Then after the episode, we update the neural network with the real reward. Equation 3.4 shows how the algorithm adjusts the network's weights using a gradient descend algorithm.

$$\underbrace{\Delta w}_{\substack{\text{Change in}\\ \text{weights}}} = \alpha[\underbrace{(R + \gamma \max_{a'} \hat{Q}(s_{t+1}, a', w))}_{\substack{\text{Maximum possible}\\ \text{Q-value for the next state}}} - \underbrace{\hat{Q}(s, a, w)}_{\substack{\text{Current}\\ \text{predicted}\\ \text{value}}}] \underbrace{\nabla_w \hat{Q}(s, a, w)}_{\substack{\text{Gradient of our}\\ \text{current predicted}\\ \text{Q-value}}} \tag{3.4}$$

The change in weights depends on the difference between the predicted value for the current state, the highest value during the next state, and the gradient of the predicted value, i.e. $\nabla_w \hat{Q}(s, a, w)$.

Experience replay [74] [75] helps to avoid forgetting previous experiences and reduces

Algorithm 3 Deep Q-learning with Experience Replay [8] [74]

Initialize replay memory $\mathcal{D}$ to capacity $\mathcal{N}$
Initialize action-value function Q with random weights
for each decision epochs **do**
  Initialise sequence $s_1 = x_1$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  With probability ϵ select a random action a_t
  otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
  Execute action a_t and observe reward r_t
  Set $s_{t+1} = s_t, a_t, x_{t+1}$ and pre-process $\phi_{t+1} = \phi(s_{t+1})$
  Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
  Sample random mini-batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
  Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a_{t+1}} Q(\phi_{j+1}, a_{t+1}; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
  Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
end for

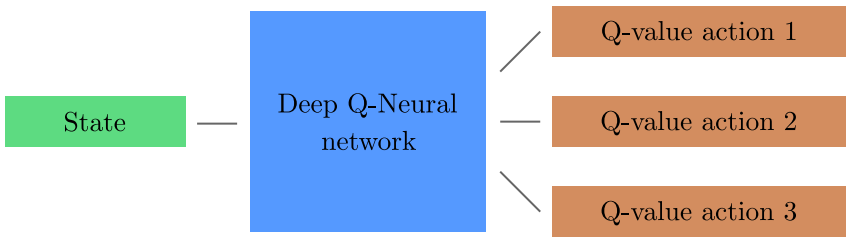

FIGURE 3.5 – Deep Q-learning

correlation between the experiences. Because of high correlation between actions and states, the weights of the network are highly variable. At each interaction with the environment, we receive a tuple (state, action, reward, new state) and use it to learn the best action to take. The problem is that this information is obtained sequentially and the network tends to forget the previous experience since it overwrites them with new experiences. With the experience replay, we decrease this problem by storing the previous experiences in a replay memory while interacting with the environment, thus the network is fed with only a small batch of experiences. Memory size has a non-monotonic effect on learning rate, too much or too little memory both can slow down learning [76]. Depending on the MDP used and the application, the size of the memory replay may be larger than the memory used with a look-up table approach.

The capacity of generalization of ANNs is a major advantage compared to look-up

tables. A neural network does not need to explore all state-action pairs. The network can find good solution without exploring the states by generalizing its knowledge. However, Deep Q-learning needs different experiences to avoid overfitting. Overfitting appears when the network fails to reliably predict future observation.

This algorithm also needs to adjust some parameters as the previous ones. The learning rate α determines how fast the new experience replaces the old ones, and a neural network is more sensitive than a look-up table to this parameter evolution since it impacts all the weights and thus all the computed rewards. So, we set α to 0.1, which is a balance between network stability and convergence speed. The value of α is determined after several experiments. The discount factor γ still represents the importance given to the future reward over the immediate one and we set it to the same value (0.8). We tested different hyperparameters for our neural network and finally, we selected a neural network composed of 1 input layer neuron, 1 hidden layer of 20 neurons, and 10 neurons in the output layer. An overly complex or simple network will not be efficient at all. The chosen activation function [1] is a rectifier (Fig. 3.6) for all layer except for the output layer, which uses a linear function. We store in a memory the last 10 experiences in order to use them as a batch.

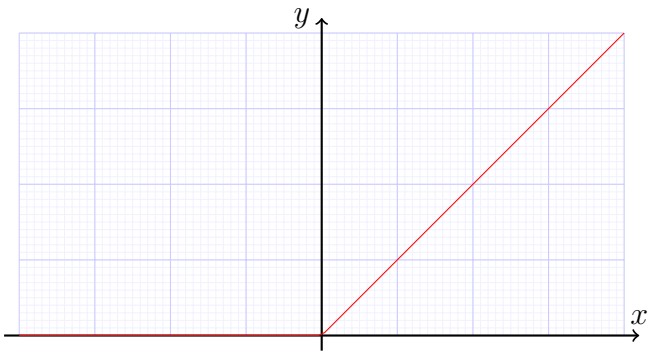

FIGURE 3.6 – Activation function: linear rectifier (Eq. 3.5) (ReLU [77])

$$f(x) = x^{+} = max(0, x) \tag{3.5}$$

Simulation results for Deep Q-learning

As for the previous algorithms, we simulated the system over a period of 21 days. Figure 3.7 displays the evolution of the battery load and the sampling frequency of the

1. For more information, the first chapter presents how a neural network works.

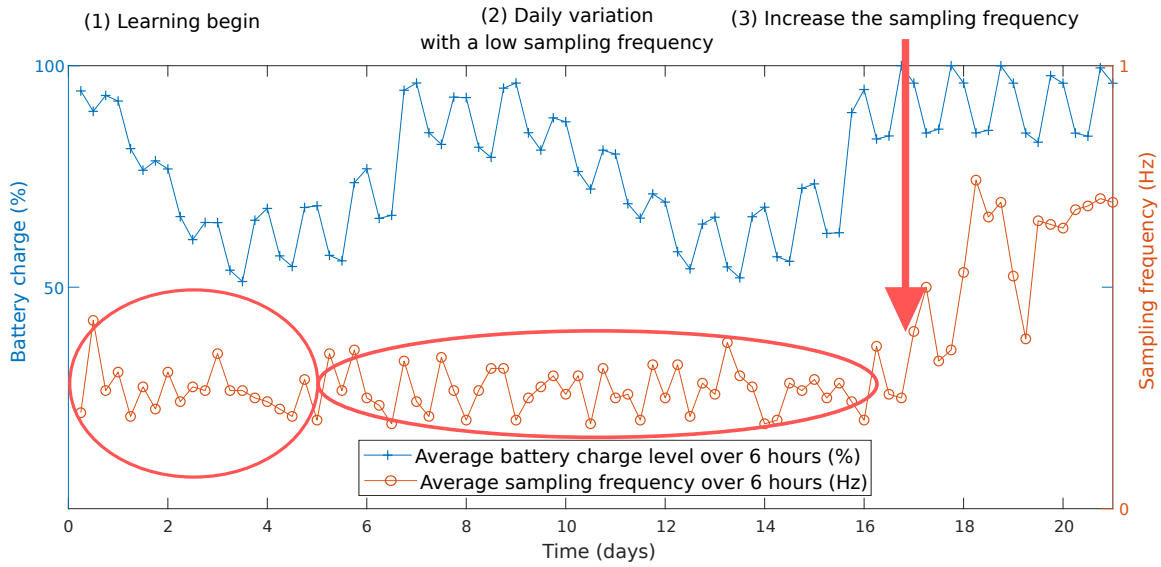

FIGURE 3.7 – Evolution of the battery charge and sampling frequency of the sensors using the Deep Q-learning algorithm

sensors. At the beginning (1) the sampling frequency is low and increases after day 17 (2). Nevertheless, we start to observe the daily variation around day 4. At the end the sampling frequency increases and the daily variation disappears. The agent loses the information it learned at the beginning.

The Deep Q-learning algorithm achieves an energy management of the node. However, it forgets over time the daily variation information. A way to improve the learning is to store only relevant information in the memory, but the question is how to determine which information is relevant.

The network employed here is based on the one used in [8]. However, different optimisations have been proposed during the last years such as Double Q-learning [78], Prioritized Double Q-learning [79], etc. [80] compares and combines all the optimisations. Nevertheless, all these optimisations are not suitable for embedded devices. Double Q-learning requires a second identical ANN to be trained independently, which increase the computational and memory needs.

3.6 Comparative results

We conducted simulations on three different reinforcement learning algorithms using the same decision process. The goal of each algorithm was to manage the energy consump-

tion of an identical marine buoy equipped with solar panels. The variation in the harvested energy during the day makes the energy management challenging. They all succeed to adapt the sampling frequency to this daily variation. These algorithms use Q-values to determine the best action to select and the convergence of the Q-values is difficult to anticipate. The algorithms have probably not reached the optimal policy but provide good enough decision to preserve the system's battery during the deployment.

The choice of the RL algorithm to use is not straightforward and depends on application requirements, computational capabilities and available memory. Moreover, there are numerous different embedded systems and their capacity in memory and processing vary a lot. Each algorithm has its own advantages and so, the choice of the trade-off will depend on the context.

In order to provide designers with guidelines, we compare the algorithms using four criteria : the computational requirement, the memory needs, the learning speed and the stability of the algorithm. The computational requirement is important for an embedded system. Indeed, these systems have limited computing capacity and often time constraint applications. Moreover, they consume energy to compute the Q-value and it would be counter-productive to consume more energy or to take more processing time for the energy management algorithm than for the main application. The Q-learning algorithm is the least processing-hungry algorithm since it computes only a value at each iteration. The Deep Q-learning computes all the Q-values at each iteration and the training of the neural network requires processing too.

Memory usage is another parameter to consider since the memory available on a microcontroller unit is often low (few kB). The neural network approach needs less memory than a look-up table for large environment since it stores only the weights of the neurons, whereas the look-up table stores all the Q-values. Moreover, it is possible to further reduce the memory usage of a neural network by reducing the number of neurons but we decrease the accuracy of the computed Q-values. For instance, the presented MDP used only 10 states to represent the battery charge level, if we want to increase the number of states the look-up table size increases, when the NN size stay the same.

The learning speed requirement depends on the application. Reinforcement learning approaches can only be used when the system can make errors safely and learn. However, some applications need to take good decisions quickly after the deployment of the node. Deep Q-learning surpasses other algorithms on the learning speed. In fact, the neural networks have the property to generalise the learning, which makes the knowledge obtained

for the Q-values in a particular state impact the Q-values computed in all the different states. In a large environment, the amount of time required to explore each state to create the required Q-table would be unrealistic. A intermediate solution to have the benefits of a good convergence speed while using a look-up table, is to implement the Dyna Q-learning algorithm and then disable the model of the environment when the learning rate α is low reducing the memory usage.

The stability depends on the impact that new information can have on the algorithm. The stability of the neural network approach is lower; indeed, with each network update, all future Q-values are modified. While a new experience with Q-learning only changes one Q-value and only once. With the Dyna Q-learning, this Q-value can be modified several times before repeating the experiment with the use of a partial model. This criterion becomes important when the learning ends because a bad experience can modify the agent's behaviour, which is why the value of α is modified as the exploration progresses, reducing this risk.

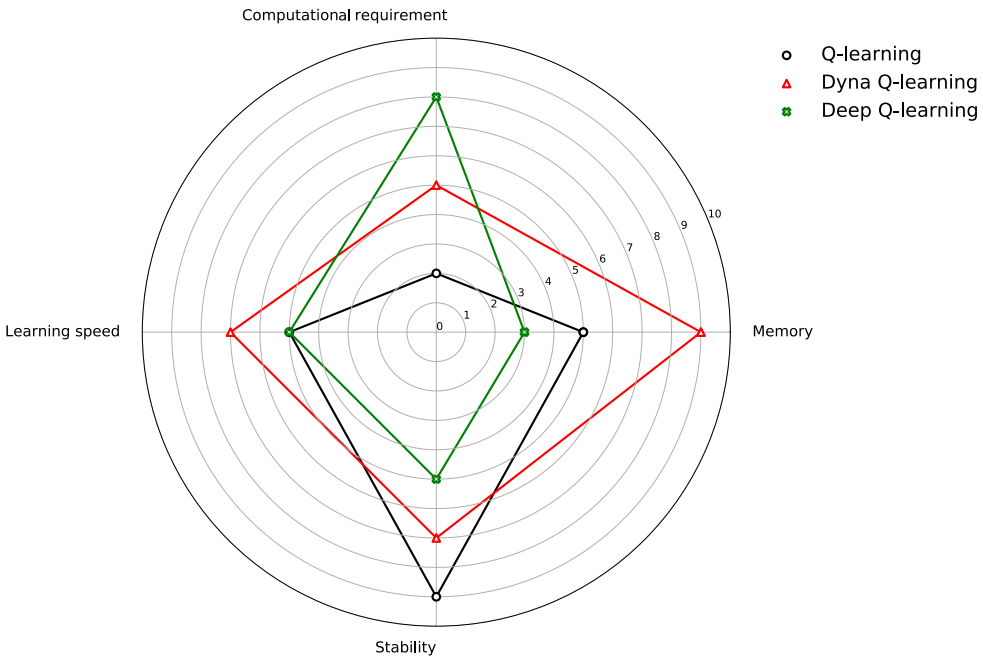

FIGURE 3.8 – Algorithms comparison in terms of memory and computational requirements, learning speed and stability

These different parameters (Fig. 3.8) should give designers the guidelines to choose the most appropriate algorithm for the energy management of the system they are developing.

After choosing the algorithm, the designers must determine the correct parameters for their application either based on experience or empirically.

Conclusion

RL approaches allow a system to adapt dynamically to changes in its environment. There is a lack of metrics to help designers choose an appropriate approach for a given application. This chapter presented a classification to help designers to select an appropriate solution depending on their system's constraints. These criteria are learning speed, stability as well as memory and computational requirements. The comparison shows that each approach has its own advantages and drawbacks. The choice of the approach depends on the application and the trade-off between computation and memory used.

However, our decision process does not satisfy completely the Markov propriety. A stochastic process has the Markov property if the conditional probability distribution of future states of the process depends only upon the present state, not on the sequence of events that preceded it. Nevertheless, the energy's consumption vary with the ageing of the component and a hardware failure such as a solar panel destruction will influence greatly the transition probability. That is why we always keep α greater than 0, so the learning never stops.

Once the choice of the appropriate algorithm is done, another effort must be done to propose an suitable reward function. There is a lack in the literature on how to design a good reward function to give the correct behaviour to our agent. In the following chapter, we will present our proposal for a methodology to help to conceive the reward function.

Chapter 4

Reward Function Design

Contents

4.1 Reward Function Evaluation 70

4.1.1 Presentation of the use case 70

4.1.2 Presentation of the decision process 72

4.1.3 Experimental Results 73

4.2 Design of a Piecewise Reward Function 77

4.3 Continuous Reward Function to Balance the Performance and the Energy Consumption 80

As explained in the conclusion of the previous chapter, RL algorithms have several parameters that must be configured. The reward function design is an important task when we use an RL algorithm. Indeed, the choice of an appropriate reward function is difficult since this function determines the behaviour of the system, choosing it is an essential task for the system designer. Still, the literature on RL algorithm rarely discusses the choice or the design of the reward function.

This chapter is organized in three parts. First, we evaluate different reward functions. The objective of these functions is to manage the energy of a body sensor node for a cardiac monitoring application. This work's goal is to identify the most suitable variables to use in order to design a good reward function for the energy management of a sensor node. In a second part, we propose a new adaptive reward function that can adjust the balance between the node's performance and its energy consumption according to the battery charge level. Finally, we improve the proposed reward function to reduce the number of parameters to be determined for the function to be as efficient as possible. This step makes the reward function more easily applicable.

4.1 Reward Function Evaluation

The work presented in this section was realised during a three-month research visit in Tallinn University of Technology (TalTech), Thomas Johann Seebeck Department of Electronics, May-June 2018.

4.1.1 Presentation of the use case

In order to evaluate the performance of different reward functions, we use a simple application of a Wireless Body Area Network (WBAN): monitoring of the cardiac activity. The objective is to manage the energy consumption of a sensor node fitted on a human chest to monitor the cardiac activity for a non-medical application (Fig. 4.1). The heart beat is measured during 10 seconds. Then, data is sent immediately to a smartphone to be processed. The smartphone is used as a gateway and communicates with the node using a Bluetooth Low Energy (BLE) transceiver. The node does not continuously monitor the cardiac activity; after each measurement it enters in a sleep mode to minimise the energy consumption. The period of sleep, between two measurements, is variable and lasts from 1 to 60 minutes.

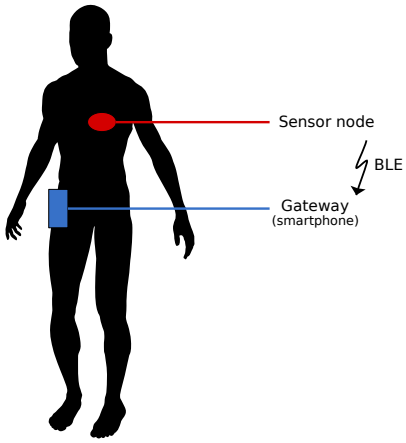

FIGURE 4.1 – Sensor node fitted on a chest to monitor the heart beat

The sensor node is equipped with an optical heart rate detection sensor, a low-power micro-controller unit (MCU), a BLE transceiver and a battery with a capacity of 100 mAh. The energy consumption of each component is summarised in Table 4.1. The energy consumption of the MCU depends on the processor clock frequency; the maximum frequency is 32 MHz, implying a maximum current consumption of 7.2 mA. As can be seen in Table 4.1, the communication unit in active mode consumes more than the two other components combined. When the node is in sleep mode, it still consumes energy except the communication unit that can be fully switched off.

Component	Active mode	Sleep mode
Heart rate sensor	1.6 mA	0.12 mA
Micro-controller	225 µA/MHz	0.93 µA
BLE transmitter	10.5 mA	0 µA (turned off)

TABLE 4.1 – Node components and respective current consumptions

A kinetic motion energy harvester is added to the system in order to increase the amount of energy available and extend the node's lifetime. This energy harvester converts the energy of the node's wearer movements into electrical energy and is presented in [81]. Although the harvested energy is low, it still can extend the node's lifespan; Table 4.2 shows how much power can be harvested according to the activity of the wearer. These data are extracted from [81].

We use the dominant frequency of motion, F_m, to identify which activity is performed by the wearer. We obtain F_m by determining the maximum spectral component of the Fourier Transform of the acceleration $a(t)$. Since the harvested energy is uncertain and

Activity	Power harvested
relaxing	2.4 µW
walk	180.3 µW
run	678.3 µW

TABLE 4.2 – Kinetic motion's power harvested for three different activities

depends on the activity of the wearer, the use of an RL approach is encouraged to manage the node's consumption by adjusting its sleep duration and its processor's clock frequency.

4.1.2 Presentation of the decision process

In this work, we compose a set of actions with different processor frequencies (F_p) and periods between each measurement (P_s) (Table 4.3). For instance, Action 1 has a processor frequency of 32 MHz and a measurement every minute, whereas Action 3 has a processor frequency of 4 MHz and a measurement every 5 minutes. Thus, Action 1 consumes more current than Action 3. All actions have different energy consumption levels since they depend on the processor's frequency in active mode and its consumption in sleep mode (see the fourth row in Table 4.3).

Action	F_p	P_s	Average current consumption	Average energy consumption
1	32 MHz	1 min	0.6278 mA	523 nJ
2	4 MHz	1 min	0.4873 mA	406 nJ
3	4 MHz	5 min	0.2292 mA	191 nJ
4	4 MHz	20 min	0.2044 mA	170 nJ
5	1 MHz	60 min	0.1926 mA	160.5 nJ

TABLE 4.3 – Set of actions with both different processor frequencies (F_p), periods between each measurement (P_s), and the associated average current consumption

The state space is divided into three different parts corresponding to the activity of the wearer (Table 4.2). We use the dominant frequency motion F_m, which is correlated with the energy the node harvests to consider our state; a high value of F_m corresponds to more energy being harvested and a low value of F_m corresponds to less energy being harvested. The current state is identified with the value of F_m and corresponds to an activity. The activity can be considered high (i.e. running) if $F_m > 2$ Hz , moderate (i.e. walking) if 2 Hz $\geq F_m > 1$ Hz or low (i.e. relaxing) if $F_m \leq 1$ Hz.

4.1.3 Experimental Results

First of all, it should be noted that the harvesting capabilities of the kinetic motion harvester are not sufficient to fully recharge the sensor node's battery. So we seek and expect to reduce the node's consumption when the harvested energy is low. We test five different reward functions to identify which parameters have a significant impact on our system's behaviour. To avoid divergence in the Q-values, the values of the different reward function are bounded to [-1, 1].

There are different constraints when designing the system and most of them are conflicting; for instance maximizing the number of measurements while also reducing energy consumption. The main purpose of the RL algorithm is to find the equilibrium point to respect these constraints. To this end, the first and second reward functions use a parameter β to balance the equilibrium point according to what is considered most important between performance and battery level [53].

The first reward function (R1) seeks to balance the conflicting objectives between the sleep duration P_s and the energy consumption of the sensor node. $B_r(t)$ is the residual energy in the battery's node at time t.

$$R = \beta * \frac{\min(P_s)}{P_s} + (1 - \beta) * (B_r(t) - B_r(t-1)) \tag{R1}$$

The second reward function (R2) is similar to the first one but instead of using the energy consumption, it only uses the residual energy of the battery's node at time t.

$$R = \beta * \frac{\min(P_s)}{P_s} + (1 - \beta) * \frac{B_r(t)}{B_{max}} \tag{R2}$$

The third reward function (R3) does not consider the sleep duration P_s but only the energy consumption. The objective is to find the less consuming operating mode without taking care of the performance.

$$R = B_r(t) - B_r(t-1) \tag{R3}$$

Finding the right setting for β is not trivial, that is why the fourth reward function (R4) uses the product of the sleep duration P_s and the residual energy $B_r(t)$. Indeed, the

result is maximum when both values are maximum.

$$R = \frac{\min(P_s)}{P_s} \times B_r(t) \tag{R4}$$

The primary goal is to adapt the energy consumption to the activity of the wearer. So, in the fifth reward function (R5), we use the dominant motion frequency F_m which determines the activity. The aim is to minimise the difference between the normalised F_m and the energy consumption; a cosine function restricts the reward to $[-1, 1]$. The reward is maximised when the difference is near to 0. Moreover, this reward function eliminates the β parameter that is not trivial to adjust. $\mathcal{N}$ is the rescaling function and consists in rescaling the range of features to scale the range in [0, 1].

$$R = \cos\left(\frac{\mathcal{N}(F_m) - (B_r(t) - B_r(t-1))}{2}\right) \tag{R5}$$

We simulate a week of node's deployment to observe the evolution of the battery's charge level. The activity changes every 30 minutes, and the agent chooses an action (Table 4.3) every 20 minutes. The activity and the decision change at different times, thus the energy harvested has changed when the agent chooses an action. Moreover, the energy harvested fluctuates around 20% the values of Table 4.2. Figure 4.2 shows the average energy consumption of the node according to the activity identified with the dominant frequency of motion, F_m. The parameter β is fixed at 0.3 since our primary goal is to adapt the node's consumption, i.e. we give more importance to the energy factor. The results show that the choice of the reward function has a significant impact on the average current consumption; while some reward functions yield the expected behaviour, others adapt poorly to the wearer's activity and others do not yield the correct behaviour at all, as discussed in what follows.

The expected behaviour of the node is to adjust its energy consumption depending on the harvested energy. Thus, during a physical effort, when the harvested energy is high, the node realises more measurements. A second objective is that the node needs to survive at least a week to reduce the number of human intervention to recharge the node. This second objective is achieved for all the reward functions.

▶ Reward function R1 computes the reward using the sleep time and the energy consumption of the node. This function produces a maximal value when the sleep time and the energy consumption are low. It successfully adapts the energy consumption according

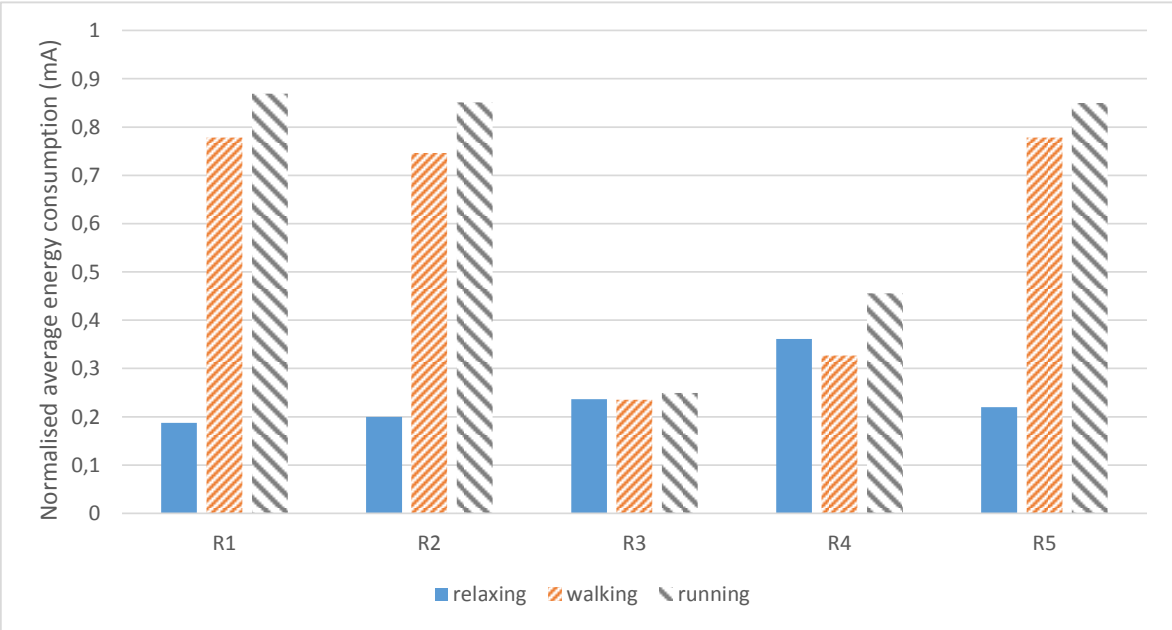

FIGURE 4.2 – Normalised average energy consumption of the node according to 1) the activity of the wearer and 2) the reward function used within the Q-learning algorithm. R1, R2 and R5 behave as expected since they allow the node to consume more when more energy is harvested.

the activity, increasing the node's consumption when the harvested energy increases. A drawback may be how to best choose the β parameter value.

▶ Reward function R2 computes the reward with the sleep time P_s and the battery's residual energy. In the same way than the reward function R1, it successfully adapts the energy consumption according the activity, achieving lowest energy consumption as the reward function R1 in 2 activities (walking and running). Furthermore, both reward functions share the same drawback, i.e., the choice of the value of the β parameter.

▶ Reward function R3 computes the reward only using the node's consumption. It fails to adapt the node's behaviour according to the harvested energy. It does not make any difference between the activities; however, it succeeds to minimise the energy consumption. At the end of the simulation, the battery charge is still above 75%. Nevertheless, the frequency measurements is the same regardless of the activity.

▶ Reward function R4 computes the reward with the product of P_s and the battery's residual energy. This reward function does not have a parameter to tune, and it is easy to compute. Unfortunately, it fails to adjust the node's consumption according to the

harvested energy. The reward function increases the node's consumption when the wearer is relaxing (i.e. the energy harvested is low), decreases it when the wearer is walking and then increases the node's consumption when the wearer is running (i.e. the energy harvested is maximal). This is obviously not desired and is due to the fact that reward function (R4) is more influenced by the sleep time P_s than by the consumption of the sensor node.

► Reward function R5 computes the reward with the normalised value of the dominant frequency of motion F_m and the node's consumption. The reward is maximal when the difference between the energy consumption and the normalised dominant frequency of motion is close to 0. The reward function R5 fulfils the different objectives we had set at the beginning. As there is no parameter to tune, this reward function can be used easily in this application. Nevertheless, the absence of this parameter makes this function less appropriate for other applications with different requirements.

Reward function	Configurable	Energy consumption	Compliance with the objectives
R1	★★★	★★	★★★
R2	★★★	★★	★★★
R3	–	★★★	–
R4	–	★	–
R5	–	★★	★★★

Table 4.4 – Evaluation of the different reward functions

The overall evaluation of the five reward functions is summarised in Table 4.4. Reward functions R1 and R2 allow to regulate the importance given to the energy consumption according to the application requirements by increasing or decreasing the value of β, whereas reward functions (R3) and (R4) are not relevant to adapt correctly the energy in a sensor node. The correct behaviour of the node can be obtained by using a β parameter to balance energy consumption and performance (R1, R2). However, it is necessary to adjust this parameter. Using the dominant motion frequency in R5 removes this parameter and still achieves the right behaviour. However, this reward function is less modular. It allows adapting the energy consumption according to the activity but does not take the sleep duration into account.

In this section, we have experimented with different reward functions in a series of simulation to identify the best design. We find out that including a balancing parameter to adjust the trade-off between performance and energy consumption is a good solution.

However, this design requires a tuning of its parameter. The best tuning is found by an expert or empirically. In the following sections, we present two reward functions designed to adjust the balance between the battery charge level and the performance depending on the battery charge level.

4.2 Design of a Piecewise Reward Function

As seen in the previous section, the use of a balancing parameter allows a designer to design a reward function that complies with the objectives. Moreover, the use of a balancing parameter makes the reward function configurable, either to maximize a performance parameter or to preserve the battery' energy. In this way, we propose a reward function in which the configuration changes depending on the battery level (Equation 4.1).

$$R = \begin{cases} F_s \times \rho_1 + B \times (1 - \rho_1) & B \geq 75\% \\ F_s \times \rho_2 + B \times (1 - \rho_2) & 75\% > B \geq 50\% \\ F_s \times \rho_3 + B \times (1 - \rho_3) & 50\% > B \geq 25\% \\ F_s \times \rho_4 + B \times (1 - \rho_4) & \text{otherwise} \end{cases} \tag{4.1}$$

where $1 \geq \rho_1 > \rho_2 > \rho_3 > \rho_4 \geq 0$, F_s the sampling frequency and B is the charge of the battery.

When the battery is fully charged, it is useless to preserve it and it possible to maximize the performance. Whereas, when the battery is discharged, it becomes really important to restrict the energy consumption in order to recharge the battery and extend the node's lifetime. A difficulty with this reward function, is its adjustment. In Equation 4.1, the battery is divided into four equal-sized parts; however, this may be different according to the application or the node. Moreover, the parameters $\rho_{1,2,3,4}$ must be selected and a poor choice will make the reward function less effective or even not effective at all.

To evaluate the proposed reward function, a simulation is conducted for the deployment of a buoy near Lorient, France, during 21 days. The buoy use case presented in Chapter 3 is preferred instead of the body sensor node. Indeed, the body sensor node does not harvest enough energy to recharge the battery, and the reward function in Eq. 4.1 requires to have different battery charge cycles for the agent to converge. The Q-learning algorithm is applied with the proposed reward function (Eq. 4.1). The value of the learning rate α is computed using the same equation as previously (Eq. 3.3), and we set the value of the discount factor γ to 0.8. The agent chooses an new action every 30

minutes to let the battery change.

Several experiments have been conducted to find the most suitable values for the parameters $\rho_{1,2,3,4}$. We found out that the best values are: $\rho_1 = 1$, $\rho_2 = 0.6$, $\rho_3 = 0.3$, $\rho_4 = 0$. Thus, when the battery is more than 75% charged, the reward is computed only with the frequency of measurements. When the battery level is between 75% and 50%, the reward is computed with both frequency and battery level, but the frequency has more importance. It is the opposite when the battery level is between 50% and 25%, the reward start to preserve the battery's energy when the energy harvested is not sufficient to recharge it. If the battery charge level decreases below 25%, the reward is computed only with the battery level in order to preserve the node's energy. The results of a simulation using these values for the reward function are presented in Figure 4.3.

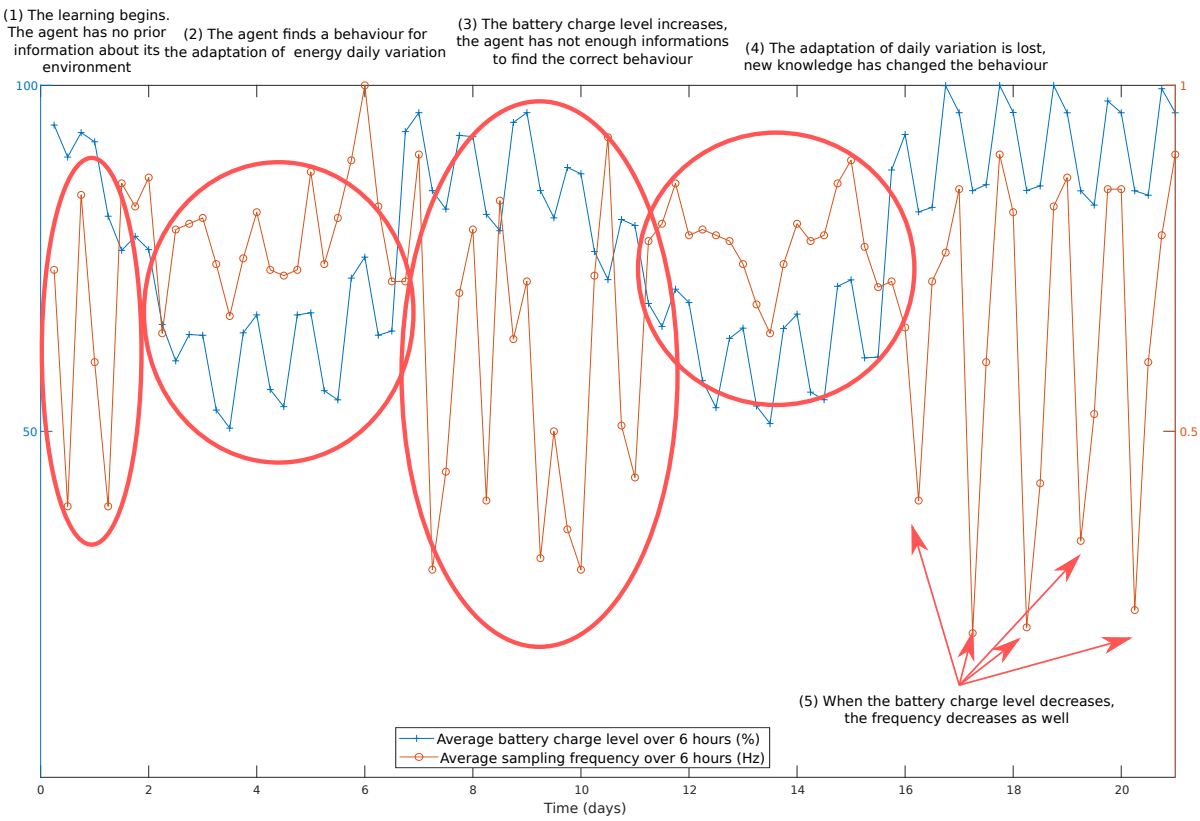

FIGURE 4.3 – Evolution of the battery charge level and frequency measurements of sensors using the Q-learning algorithm with the reward function of Equation 4.1

At the beginning of the deployment (1), the agent has no prior information about its environment and takes random actions to explore it. When the battery decreases (2), the agent adapts the frequency of measurements and finds a behaviour which adjusts the

frequency according to the daily variation in the battery charge level. When the battery charge level increases around day 6 (3), the agent has not enough information in the new state to find a correct behaviour. And when the battery decreases again (4), the first knowledge learned has been replaced with the new information; the agent lost the behaviour which adjusts the frequency of measurements according to the daily variation. Nevertheless, at the end of the simulation (5), the agent's behaviour adapts correctly the frequency to the variation, this behaviour receives enough rewards to be reinforced.

The battery does not decrease enough, the agent never explores the environment when the battery level is critical. So a second experiment is conducted where the battery capacity is reduced to 3.2 mA/h instead of 5.2 mA/h. The result of this simulation is shown in Figure 4.4.

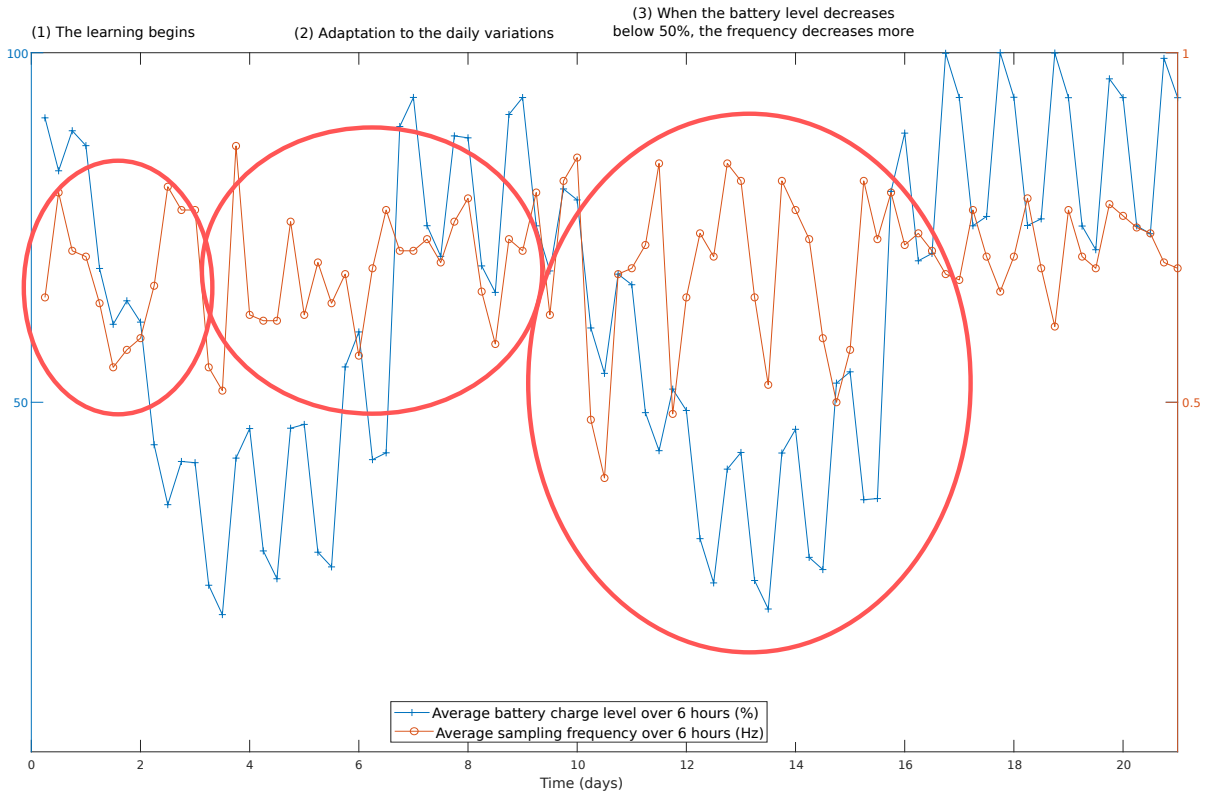

FIGURE 4.4 – Evolution of the battery charge level and measurements frequency of sensors using the Q-learning algorithm with the reward function 4.1

At the beginning of the simulation (1), the agent has still no prior information about its environment. Then, it successfully adapts to the daily variations in the battery charge level (2). And when the battery level decreases below the 50% of charge level (3), the agent

decreases the frequency of measurements more than during the previous simulation. The results show that the proposed reward function is suitable to preserve the battery and to maximize the performance when the battery is recharged.

Nevertheless, a difficulty with this reward function is to determine the values of the ρ parameters. The adjustment of the parameters is based on the expertise of the designer or is determined empirically. To obtain the best behaviour with the use case, the parameter ρ used was determined empirically, but it can be different according to the application and the prevalence of the performance or the energy. For such an approach to be accepted, it is important to avoid to add extra-parameters to tune; in such case the reward function becomes more complex and the behaviour of the agent becomes less predictable. Thus, in the following section, we present an improvement of the proposed reward function where the parameters $\rho_{1,2,3,4}$ are removed and the different levels too. This improvement eliminates the adjustment step, making it easier to use the new reward function. Furthermore, the performance of the new reward function is similar to that of the previous one.

4.3 Continuous Reward Function to Balance the Performance and the Energy Consumption

The parametrization of the different variables used in a RL approach is time-consuming and add more complexity. So, the designer needs to reduce the number of parameters to be tuned in the reward function. The previous reward function is efficient and accelerates the learning speed compared to the reward function used in Chapter 3 (Eq. 3.1). The values of the different parameters can be adjusted to correspond perfectly to the desired behaviour for a given application. Nevertheless, most sensor nodes are used for environmental monitoring applications, and the main objective is to extend the node's lifetime. So, in this section, we present a reward function that reinforces the same behaviour as the previous one, but without any parameter to tune.

Indeed, while experimenting different values for ρ, we observe that this parameter's value varies as the battery level. Using this observation, we design a new reward function without parameter to tune (Eq. 4.2) to balance the battery charge level and the performance:

$$R = F_s \times B + B \times (1 - B) \tag{4.2}$$

where F_s is the frequency of the measurements and B the battery level. The parameters

$\rho_{1,2,3,4}$ have been replaced by the value of the battery charge level.

This reward function is a generalization of the previous one. The reward is computed mainly with the frequency of measurements when the battery is fully charged, and mainly with the battery level when the battery level is low. Thus, this proposed reward function requires no additional parameter to adjust.

We conducted a simulation on the marine buoy use case [1]. We simulated the deployment of the buoy during a period of three weeks near Lorient, France. We applied the Q-learning algorithm with the same value for α (Eq. 3.3) and $\gamma = 0.8$ (as in both Chapter 3 and the previous section).

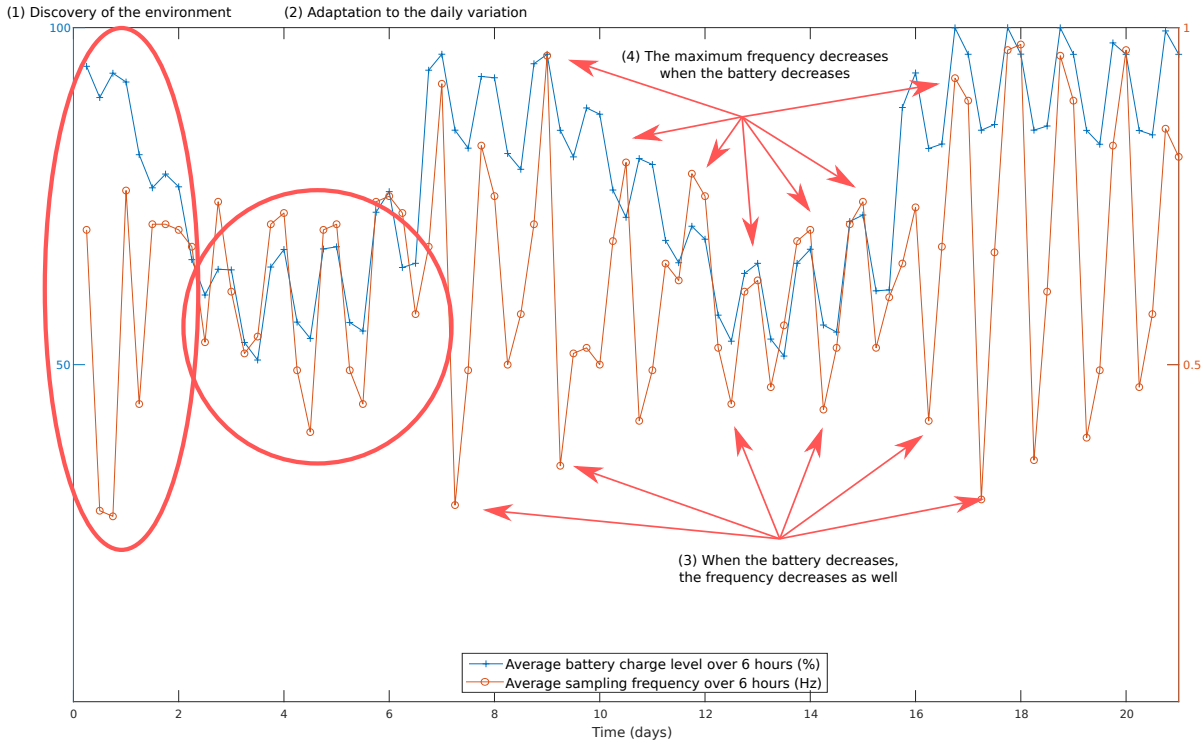

FIGURE 4.5 – Evolution of the battery charge level and measurements frequency of sensors using the Q-learning algorithm with the reward function in Eq. 4.2

The simulation results (Fig. 4.5) show that the agent adapts correctly the measurements frequency to the battery's behaviour. At the beginning of the deployment, the battery level decreases quickly and the agent adjusts almost immediately the frequency of measurements. Then, when the battery level increases due to the harvested energy, the agent reacts and increases the frequency of measurements as well. The frequency of

1. presented in Chapter 3

measurements is maximum when the battery is fully charged. Before the end of the simulation, the agent achieves the correct behaviour. The agent is able to adjust the frequency of measurements to the battery charge in less than three days, when it needs more than two weeks to adapt in the previous experiments.

The proposed reward function is able to improve greatly the learning speed. Furthermore, it adjusts the balance between the performance and the battery level according to the battery level without any balancing parameter. A second experiment is done with a smaller battery in the same way than for the previous reward function. The results of this experiment are shown in Figure 4.6.

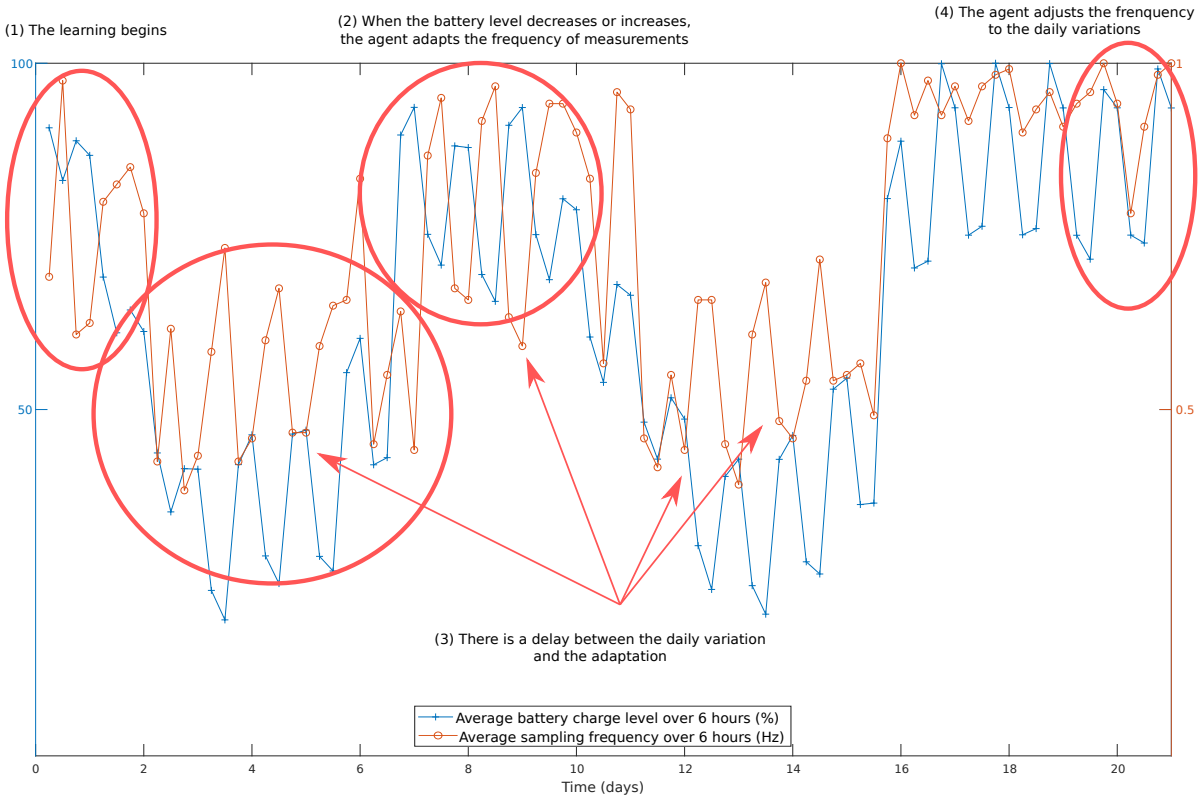

FIGURE 4.6 – Evolution of the battery charge level and measurements frequency of sensors using the Q-learning algorithm with the reward function in Eq. 4.2

At the beginning of the deployment (1), the agent has no prior information about the environment. However, when the battery charge level decreases (2) the agent correctly adjusts the frequency of measurements, even if the daily variation are not respected (3). Then, the battery charge level increases due to the energy harvested and the agent increases the frequency as well. At the end of the simulation (4), the agent seems to res-

pect the daily variation in the battery. To confirm the convergence of Q-values with the proposed reward function, we conducted a slightly longer simulation of 24 days.

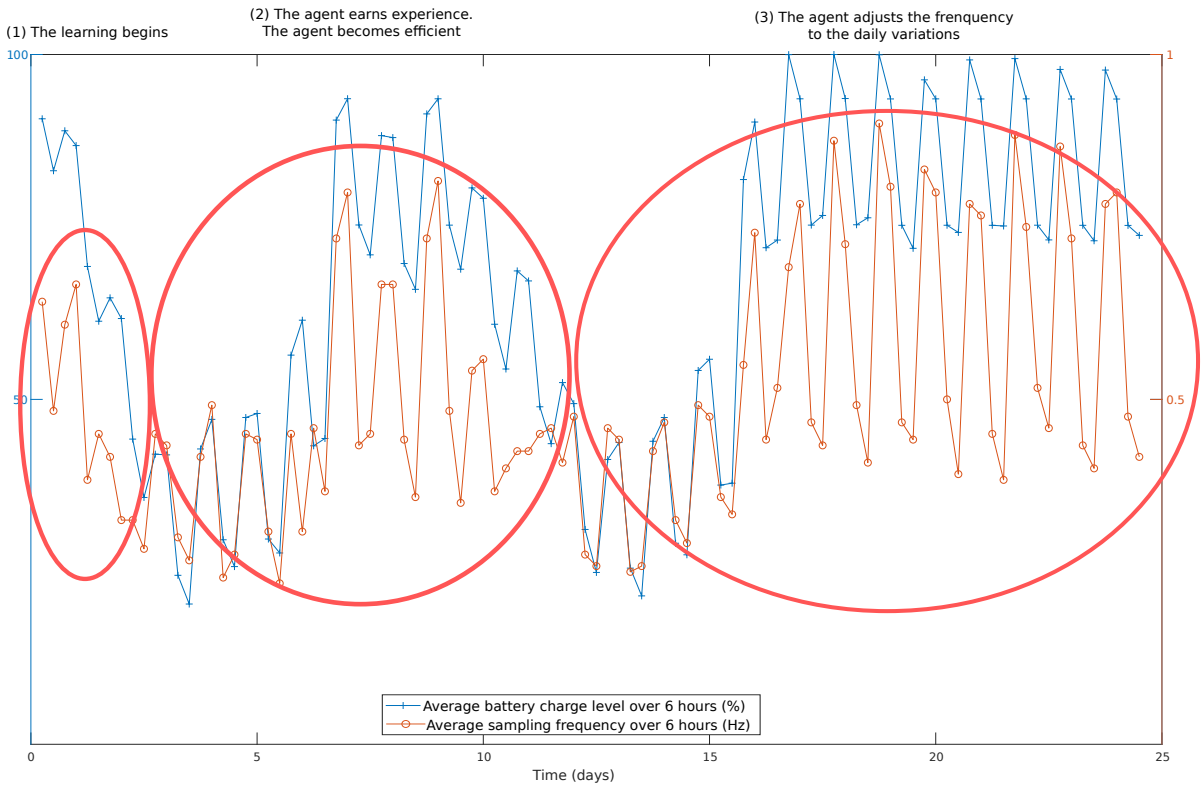

FIGURE 4.7 – Evolution of the battery charge level and measurements frequency of sensors using the Q-learning algorithm with the reward function 4.2

The simulation confirms the convergence of the Q-values. Furthermore, the agent ends up complying with the daily variation. The difference between the two simulations are due to the exploration of the environment. Indeed, during the exploration, the agent takes random actions but the agent's policy converges to the same behaviour.

When compared to the previous reward function, we observe that the new proposed reward function adapts more efficiently the frequency to the battery charge level. This reward function improves the performance when the battery charge level is high. Another important point is that the reward functions in Eq. 4.1 and Eq. 4.2 are both suitable regardless the battery capacity. The system is seen by the agent as a black box, it does not need to know the different components.

For our application, the performance parameter is the frequency of measurements, but it may be different depending on the application. The reward function should work

regardless of the performance parameter used. Indeed, better performance implies higher energy consumption.

Conclusion

The comparison of different reward functions shows that the reward function is a key component in the performance of RL algorithm. The reward function affects greatly the learning capabilities and the learning speed. Nevertheless, there are too little details about any design methodology that would provide guidelines of the reward function in the literature. A non-expert will have difficulties to use a RL approach for an energy management problem. In this chapter, we compared different reward functions to identify a efficient way to design it. We found that the use of a parameter to balance the performance parameter and the battery level works well. And it allows the designer to set more weight for a performance criterion or to save battery power.

We proposed two reward functions to balance the performance parameter of the node and the battery charge level. The objective is to preserve more the battery's energy when the battery charge level is low and to maximize the performance when the battery is fully charged. The first proposed reward function uses several functions depending on the battery charge level. The functions are similar except for the value of the balancing parameter. The function used in the simulation has four different levels (i.e. different functions). The results show that the agent is able to adjust the frequency of the measurements according to the battery charge level. The main drawback of this reward function is the different parameters to tune. Since there is no existing solution to determine the best values immediately, these parameters' values are chosen empirically or based on the expertise of the designer.

The observation of the first reward function experiments highlight the fact that the balancing parameter decreases when the battery charge level declines. Thus, we proposed a second reward function where the parameters to tune are removed, which makes it simple use. And the agent acquires a correct behaviour more quickly than with the previous reward function. Moreover, when the battery charge level decreases, the agent decreases more efficiently the frequency of measurements. Since there is no parameter to adjust in the reward function, it is easier to use the propose function. The main drawback is that this reward function is less customizable to be suitable for all different applications. Nevertheless, it complements naturally the first proposed reward function.

Another important point is that an experiment with a smaller battery capacity shows the same results; the proposed reward functions work independently of the node's components.

A limitation with the single agent approach in RL is the size of the MDP. Indeed, it is really hard to train a single agent on a large MDP, one needs a lot of data even if using NNs. And to control the energy consumption with a finer grain, one needs more precise actions, which increases the MDP. A solution is to decompose the problem into smaller problems. A single agent can be decomposed into several agents with smaller MDP which makes the RL approach more scalable. In the following chapter, we explore a new approach for the energy management of a sensor node. We apply a multi-agent reinforcement learning to control independently the different task done by the node. A sensor node has several sensors with different energy consumption which have an influence variable on the battery. So, we propose a new algorithm to adjust separately the frequency of measurements for each sensors according to the energy consumption of each.

Chapter 5

Multi-agent Reinforcement Learning Approach

Contents

5.1 Introduction to multi-agent systems 88

5.1.1 Multi-agent learning 90

5.1.2 Hysteretic Q-Learning 91

5.2 Decentralized energy management 91

5.2.1 Independent Management of Sensors 93

5.2.2 Results of the proposed algorithm with the marine buoy use case 94

In the previous chapters, we have presented different approaches in RL for energy management and we proposed a reward function to adjust the performance dynamically depending on the battery charge level. We have seen in Chapter 1 that single agent approaches are limited by the size of the MDP. A too large decision process slows the learning speed and makes its use in embedded systems difficult. A possible solution uses NNs to overcome this issue, but it needs higher processing capabilities. Another solution is to divide a complex problem modelled with a large MDP into several smaller problems modelled with small MDP and to solve independently these problems. A smaller MDP reduce the time required to explore all actions. Thus, in this chapter, we explore a different solution as compared to previous chapters, i.e., a Multi-Agent Reinforcement Learning (MARL).

A sensor node usually has different sensors with different energy consumptions. When the battery is being discharged, the reduction of the measurement of each sensors have different impacts on the overall energy consumption of the node. Logically, we want to control the measurement frequency of each sensor independently, and be able to reduce sooner the sampling of the most consuming sensors in order to preserve the battery. A MARL approach is well suited to deal with such an issue, by dividing the whole problem into smaller ones.

The first section presents multi-agent systems and the decentralized learning. Then, we present a decentralized energy management approach for a sensor node.

5.1 Introduction to multi-agent systems

Before looking at MARLs, let's introduce multi-agent systems. A Multi-Agent System (MAS) is composed of multiple interacting agents in a shared environment (Fig. 5.1). Each agent owns limited information and problem solving capacities; nevertheless, the local actions taken by the agents affect the global state of the system environment.

The applications of these systems are varied: supply management, swarm robotics, network routing, assembly line control, transportation, to even economical and medical domains. Even complex monolithic systems such as traffic lights controlling system [82] [83] [84], can be broken down into a MAS that organizes the individual agents, each of which solves a portion of the problem.

The MAS approach corresponds to a more natural decomposition of the problem and makes the learning more scalable. In fact, decomposing the actions and observations of a

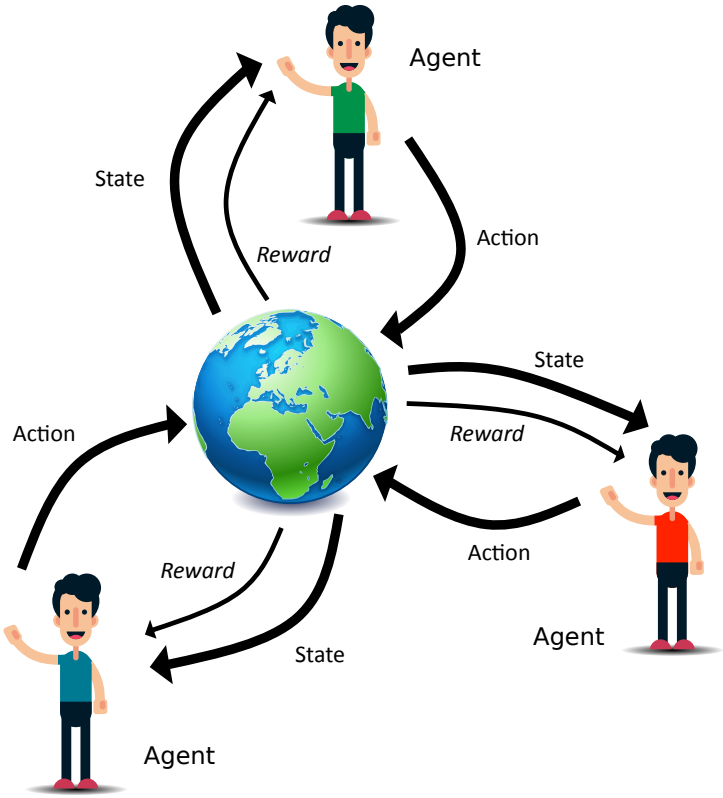

FIGURE 5.1 – Multi-Agent Reinforcement Learning in which three agents share the same environment. Each agent makes an action and the combination of all actions has an impact on the state of the environment, and each agent receives a reward depending on the state of the environment.

single monolithic agent into multiple simpler agents not only reduces the dimensionality of agent inputs and outputs, but also effectively increases the amount of training data generated per step of the environment. Moreover, a good decomposition of the problem makes the knowledge learned more transferable across different variations of an environment, i.e., in contrast to a single super-agent that may over-fit to a particular environment.

An overview of traditional single-agent RL was provided in the first chapter and describes how an artificial agent can learn an optimal behaviour policy by interacting with an unknown environment. The primary challenge in MARL is that the RL agents must consider the actions of the other participating agents in order to learn their policies and solve the problem successfully. In the following subsections, we present an overview on multi-agent learning in RL, and a state-of-the-art MARL algorithm, called hysteretic Q-learning.

5.1.1 Multi-agent learning

The generalization of the Markov decision process to the multi-agent case is the *stochastic game*, also called Markov game. A stochastic game (SG) is a tuple $\langle n, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ where:

- n is the number of agent;
- $\mathcal{S}$ is a state space of the environment;
- $\mathcal{A} = \prod \mathcal{A}_i$ is the joint action set, where $\mathcal{A}_i$ is the action space of an agent i;
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1[$ is a *transition function* specifying, for each state, action, and next state, the probability of that next state occurring;
- $\mathcal{R}_i$ is a *reward function*, specifying, for each state, action, and next state, the expected immediate reward for an agent i.

The joint actions of the agents determine the next state and the rewards received by each agent. If all agents receive the same rewards, the SG is fully cooperative, and all the agents have the same goal: to maximize the common return. It is then defined as a Multi-agent Markov Decision Process (MMDP). The objective of each agent is to find the optimal policy maximizing the expected sum of the discount rewards in the future.

The straightforward extension of centralized Q-Learning to SG considers joint actions in the computation of the Q-values. Thus, the update equation in a centralized view is:

$$Q(s_t, a_t^1, \cdots, a_t^n) = Q(s_t, a_t^1, \cdots, a_t^n) + \alpha \left(r_t + \gamma \max_{a_{t+1}^1, \cdots, a_{t+1}^n \in \mathcal{A}} Q(s_t, a_{t+1}^1, \cdots, a_{t+1}^n) - Q(s_t, a_t^1, \cdots, a_t^n) \right) \quad (5.1)$$

where s_{t+1} is the new state of the environment, a_t^n is the action of the agent n during the step t, α is the learning rate and γ is the discount factor.

The Q_i-table for the learning agent is smaller than a Q-table for a single-agent. But each agent has only a local view because it has no access to the actions of the others. Different algorithms have been proposed to manage the coordination between the learning agents. Among these different MARL algorithms, we find different variant of the Q-learning such as the distributed Q-learning [85] or the hysteretic Q-learning [86]. We describe the latter in what follows.

5.1.2 Hysteretic Q-Learning

The Hysteretic Q-learning algorithm works in cooperative MAS, where a team of independent learning agents try to coordinate their individual behaviour to reach a coherent joint behaviour. Hysteretic Q-learning assumes that each agent has no information about its teammates' actions.

In a MARL, the reinforcement received by an agent relies on actions chosen by the team. So, an agent can be punished because of a bad choice of the team even if it has chosen an optimal local action. Then, the agent had better chances to give less importance to a punishment received after the choice of an action which has been satisfying in the past. In this way, the Hysteretic Q-learning modifies the Q-value update according to whether the update is a reward or punishment. The update equation (Eq. 3.2) initially presented in chapter 3 is modified such that:

$$\delta = r - \gamma \max_{a_{t+1} \in \mathcal{A}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$$

$$Q(s_t, a_t) = \begin{cases} Q(s_t, a_t) + \alpha\delta & \text{if } \delta \geq 0 \\ Q(s_t, a_t) + \beta\delta & \text{otherwise} \end{cases} \tag{5.2}$$

where $\beta < \alpha$.

In the following section, we present, as an application case, a decentralized energy management approach for a sensor node, and we show that the proposed approach influences the sensing agent's policy according to the energy's consumption of the corresponding sensor.

5.2 Decentralized energy management

A sensor node is composed of different hardware modules: processing unit, sensing unit, communicating unit, harvesting unit, and so on. Furthermore, the sensing unit often has several sensors with different energy consumptions. We propose to use a decentralized learning to manage the different modules that compose a sensor node. Figure 5.2 shows a decomposition of a sensor node where there are two sensors, a buffer to store the data before their transmission and a communication module.

We have chosen to have one agent for each module. The communication agent optimises the data transmission according to the buffer load, whereas the sensing agents

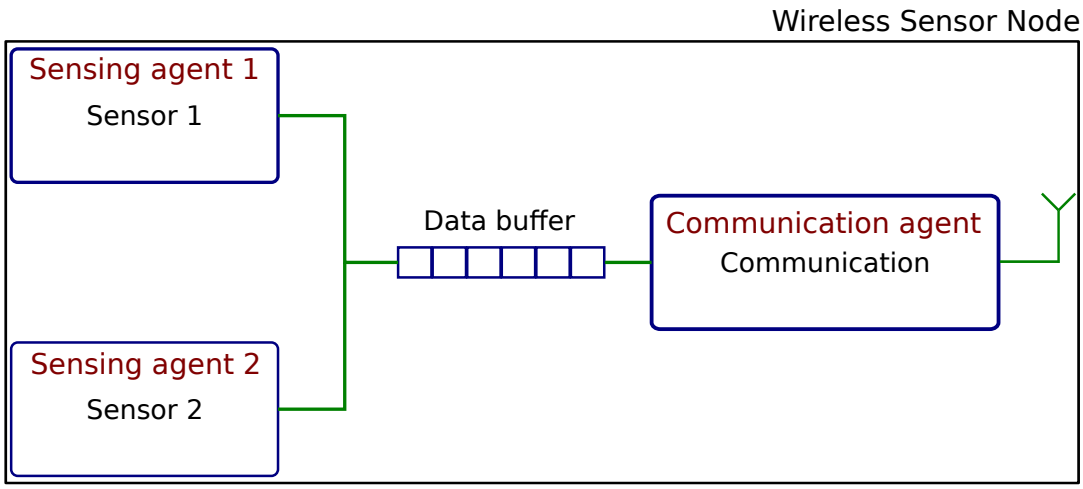

FIGURE 5.2 – Distributed Energy Management of a wireless sensor node. Two sensing agents control the measurement frequency of the anemometer sensor and the atmospheric sensor, respectively, according to the battery charge level. The measured data fill a buffer, and a communication agent controls the data transmission according to the filling of the data buffer.

adjust their sampling frequency according to the battery charge level. The environment of the communication agent and the one of the sensing agents are different. Nevertheless, each sensor has a corresponding agent, which interacts with the same environment. Their actions need to be coordinated. Reducing the sampling frequency also reduces the data transmissions, so the node overall consumption. However, the sensors do not have the same energy consumption.

In addition to use a MAS to control the different modules of the sensor node, the objective of our approach is to adapt the frequency of the measurements of each sensor according to their energy consumption, and therefore their impact on the overall energy consumption. Indeed, it is less essential to reduce the measurement frequency of a sensor with a low energy consumption than for a sensor that consumes more. Adjusting the sampling frequency of the sensors independently according to their energy consumption allows a better QoS to be maintained for a longer period of time when the battery charge level decreases. Thus, we propose a new version of hysteretic Q-learning where the β parameter, used when the agent is penalized, is adjusted according to the impact of the sensor's energy consumption on the overall consumption of the node.

In the following section, we present our approach and the results obtained on a node equipped with two sensors.

5.2.1 Independent Management of Sensors

The proposed approach uses the energy consumption of sensors to adjust the policy of the corresponding agent. So, we propose to use the following methodology to do it. The first step is to determine the impact of the energy consumption of each sensor on the overall consumption of each sensor.

1. The radio and the processor are turned off and energy harvesting is stopped; only one sensor is on and the measurement frequency is 1 Hz.
2. The variation in the battery charge level is then measured after 30 minutes.
3. Then, this sensor is turned off and we turn on the next one, and this step is repeated for each sensor.
4. We want to use the variation in the battery charge level for each sensor as a coefficient to modulate the parameter β. So the variations are rescaled between [0,1] using Equation 5.3.

$$C_i = \frac{C_i}{\sum_k C_k} < 1 \tag{5.3}$$

   where C_i is the variation measured in the battery charge level for sensor i. At the end of this step, the impact of each sensor is a value between [0,1], and the sum of all the impacts equals 1.
5. Once the previous step of the algorithm realized, we apply the hysteretic Q-learning on the sensing agents. We modified the parameter β to further penalised the most energy consuming sensors. So, the value of β is determined with the calibration value C_i obtained for the sensor i.
   The update function is modified as the follow:

$$Q_i(s_t, a_t) = \begin{cases} Q_i(s_t, a_t) + \alpha\delta & \text{if } \delta \geq 0 \\ Q_i(s_t, a_t) + C_i\beta\delta & \text{otherwise} \end{cases} \tag{5.4}$$

We test our approach with a simulation using the use case of the marine buoy [1]. We select this use case since the buoy is already equipped with two different sensors with different energy consumption. The results are presented in the following subsection.

1. The marine buoy case is presented in Chapter 3

5.2.2 Results of the proposed algorithm with the marine buoy use case

For the simulation, our system is composed of three agents (Fig. 5.2): one for each sensor and one for the communication. The data mesured by the sensors are stored in a data buffer waiting to be transmitted. The communication agent used the RL-MAC algorithm [16] presented in the Chapter 2. The reward function used for the communication agent is detailed as follows:

$$r_k(n_b, t_r) = \begin{cases} \frac{(n_s+n_r+1)\cdot T_p}{t_r-t_s} - \eta\frac{n'_b-n_b}{\sqrt{B}} & t_r, n_b \neq 0, n'_b > n_b \\ \frac{(n_s+n_r+1)\cdot T_p}{t_r-t_s} & t_r, n_b \neq 0, n'_b \leq n_b \\ -\eta\frac{n'_b-n_b}{\sqrt{B}} & t_r, n_b = 0, n'_b \neq n_b \\ 1 & t_r, n_b = 0, n'_b = 0 \end{cases} \tag{5.5}$$

where n_b and n'_b are the numbers of packets in the buffer at the beginning and the end of the frame respectively, t_r is the reserved time where the radio is active, B is the size of the buffer, T_p is the packet transmission time, n_s and n_r the number of packets sent and received during this frame, respectively, and η is a weight to modulate the penalty when the data buffer fills up.

Our sensing agents use the same MDP as the one uses for the single agent approach in the buoy use case. Thus, the state space is composed of the battery charge level by increment of 10%, and the set of actions is the frequency measurements in a range between 0.1 Hz and 1 Hz in step of 0.1 Hz. We applied the reward function proposed in the previous chapter (Eq. 4.2), and the value of the learning rate α is computed using the same equation as previously (Eq. 3.3). β is the learning rate when the agent is penalised, so its value is lower than that of α. Thus, we keep the same value as in the paper [86], which is 0.1. The value of the discount factor γ is set to 0.8, as previously.

We analysed the policy of the sensing agents in order to study the efficiency of our approach. The policy of the agent is observed in the look-up table, it corresponds to the actions maximizing the reward for each state. These values are circle in red in Figure 5.3 and Figure 5.7.

We conducted a simulation of 70 days deployment near Lorient, Brittany. Since the battery of the buoy cannot last that long, we recharged the battery entirely without delay. In this way, we avoid having the node being turned off while the battery is charging, this reduces the simulation duration. We add a penalty of -1 in the reward when the battery

is discharged, this avoids the agent to discharge deliberately the battery to receive a good reward with the immediate recharge. In realistic condition, the agent should still receive a penalty if the battery is completely discharged. Nevertheless, there will be a delay between the time the node turns off and the time it restarts because its battery has been recharged.

Our buoy is equipped with two sensors i.e., an atmospheric sensor, which consumes 7 mA, and a 3D anemometer, which consumes 55 mA. The calibration of the proposed algorithm has determined that the weight for the atmospheric sensor is 0.1129 and for the 3D anemometer, 0.8871. These weights correspond to the impact of the energy consumption of each sensor on the overall energy consumption. Figure 5.3 corresponds to the Q-table for the atmospheric sensor and Figure 5.7 corresponds to that of 3D anemometer.

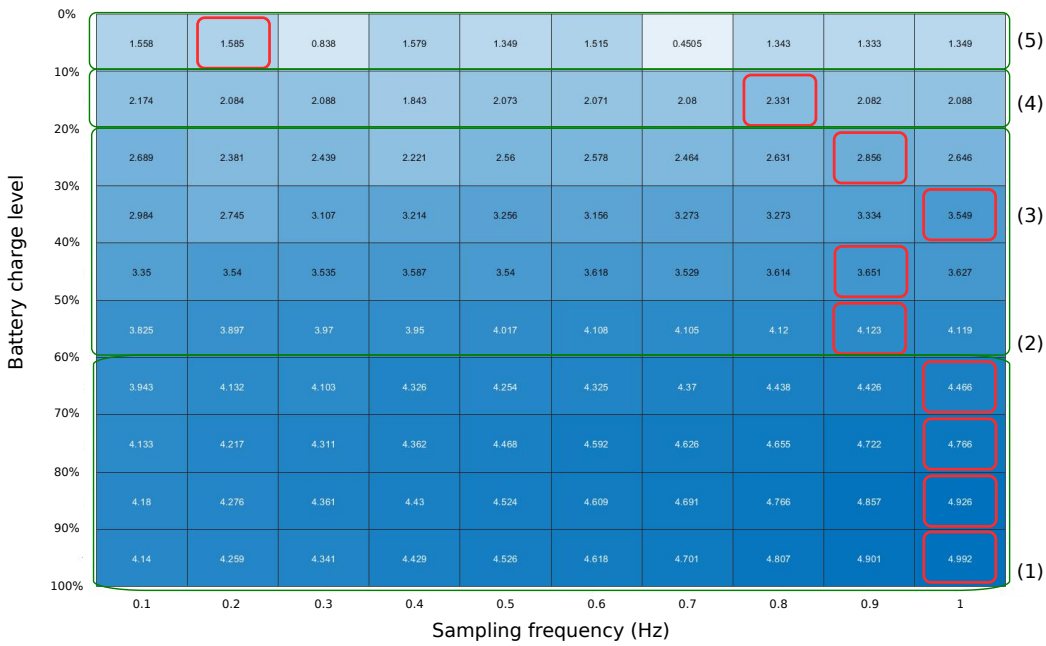

FIGURE 5.3 – Q-table of the agent for the atmospheric sensor

In Figure 5.3, we can see the Q-values which are based on the immediate reward, obtained after the taken action, and the expected reward, depending on the discount factor γ. The expected reward corresponds to the higher Q-value in the new state. We can observe that when the battery charge level decreases, the Q-value decreases as well. Thus, the agent will try to select actions that increase the battery level to increase the expected reward. The three following figures correspond to different part of Figure 5.3.

In Figure 5.4, when the battery charge level is between 100% and 60% (1), the agent's

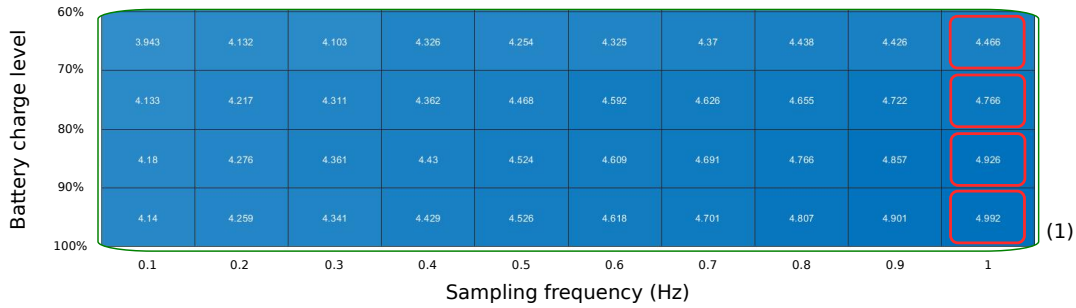

FIGURE 5.4 – Q-table of the agent for the atmospheric sensor for battery charge level between 100% and 60%

policy always select the action that maximises the measurement frequency.

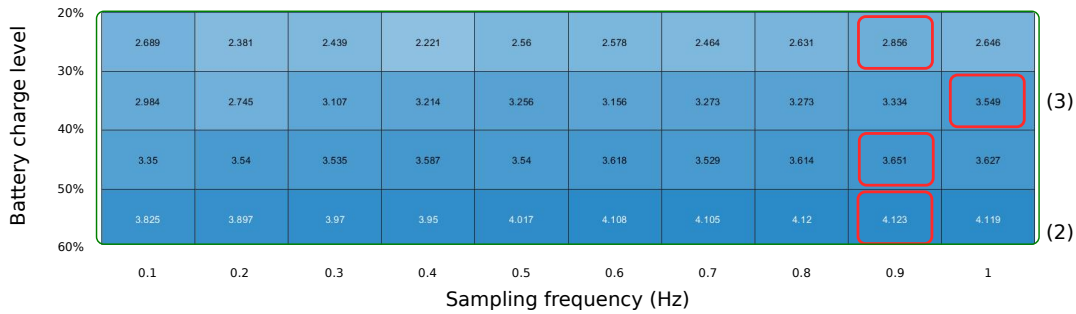

FIGURE 5.5 – Q-table of the agent for the atmospheric sensor for battery charge level between 60% and 20%

Then, between 60% and 20% (2) (Fig. 5.5), the agent chooses the action which fixes the sampling frequency to 0.9 Hz. The reward gives more importance to the battery than previously. Except between 40% and 30% (3), the action chosen by the agent is to set the measurement frequency to 1 Hz, due to the fact that the convergence of the Q-values is not totally complete.

Between 20% and 10% of battery charge level (4) (Fig. 5.6), the most rewarding action is to set the sampling frequency to 0.8 Hz. For the last 10% (5), the action selected is to fix the measurement frequency to 0.2 Hz. As we can see in the evolution of the most rewarding action in the different states, when the battery charge level decreases, the selected action decreases the frequency measurements as well , and thus, the energy consumption.

In Figure 5.7, we can observe the same trend of Q-value for the anemometer sensor depending on the battery charge level as in the Q-table for the atmospheric sensor. The three following figures correspond to different part of Figure 5.7.

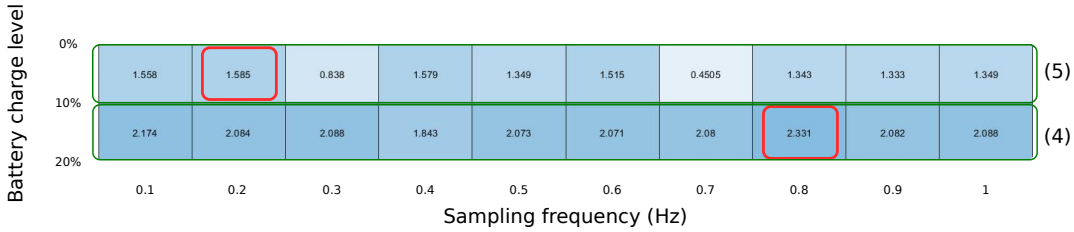

FIGURE 5.6 – Q-table of the agent for the atmospheric sensor for battery charge level between 20% and 0%

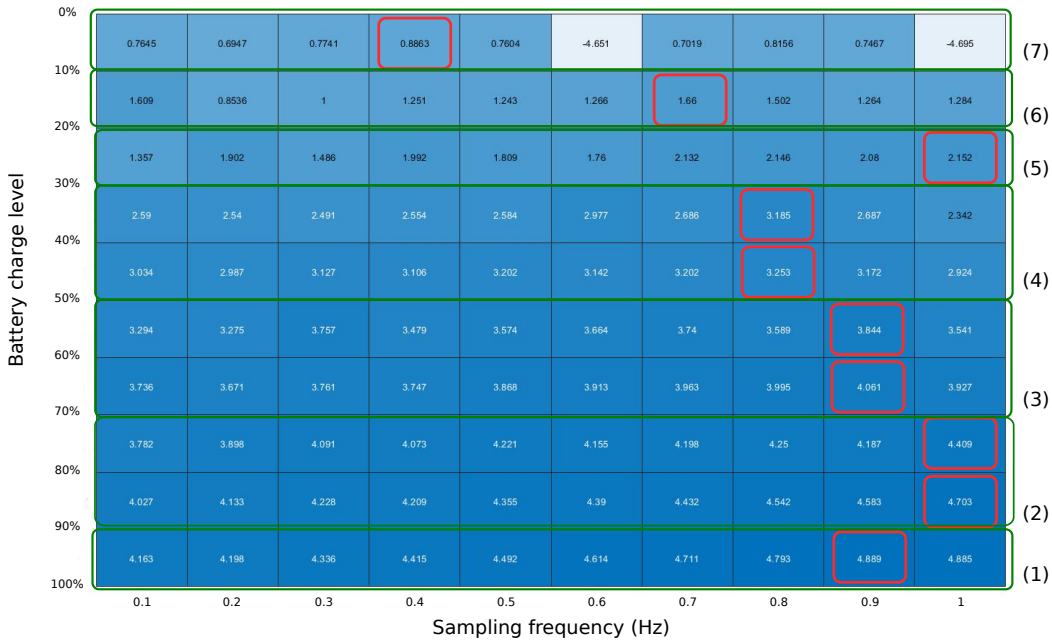

FIGURE 5.7 – Q-table of agent for the anemometer

In Figure 5.8, when the battery charge level is between 100% and 90% (1), the selected action fixes the measurement frequency to 0.9 Hz. Then, between 90% and 70% (2), the preferred action is to set the sampling frequency to 1 Hz.

Between 70% and 50% (3) (Fig. 5.9), the most rewarding action is to set the measurement frequency to 0.9 Hz. And between 50% and 30% (4), the picked action is a sampling frequency to 0.8 Hz.

The selected action increases the measurement frequency to 1 Hz between 30% and 20% (5) (Fig. 5.10), due to the unfinished convergence of the Q-values. When the battery charge level is between 20% and 10% (6), the picked action is a sampling frequency to

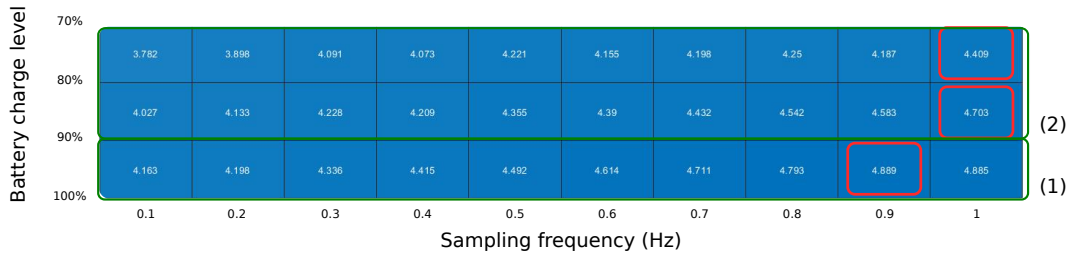

FIGURE 5.8 – Q-table of agent for the anemometer for battery charge level between 100% and 70%

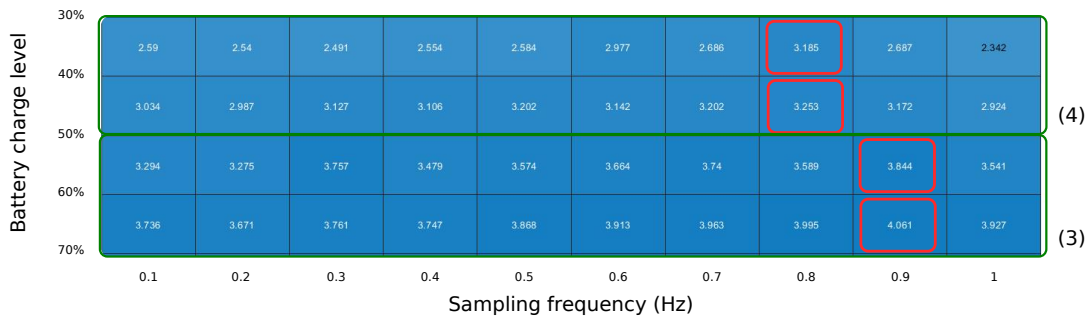

FIGURE 5.9 – Q-table of agent for the anemometer for battery charge level between 70% and 30%

0.7 Hz. Finally, between 10% and 0% of charge (7), the selected action is to fix the frequency to 0.4 Hz. As we can see in this figure, when the battery charge level decreases, the most reward action decreases as well. And as compared to the previous policy, we can see that it start decreasing sooner and faster.

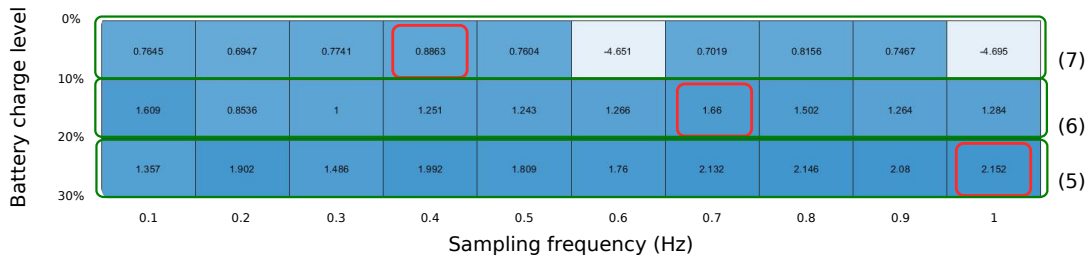

FIGURE 5.10 – Q-table of agent for the anemometer for battery charge level between 30% and 0%

Both agents decrease the measurement frequency when battery charge level decreases. This is the behaviour we already observed with the use of a single agent with the same reward function. Nevertheless, we also observe that the sensor with the higher energy

consumption start to decrease the measurement frequency sooner than the second sensor. Furthermore, in addition to a finer control of the energy consumption, the main difference between this approach and a single agent approach is the memory requirement gains. In fact, we increase a little the computations but we reduces the size of Q-tables of the sensing agents. In the single agent approach, the look-up table size is $10 \times 10 \times 10$ to control both sensors, whereas in the proposed approach the cumulative size of the look-up tables is $2 \times 10 \times 10$, which represents a decrease of 80% in memory requirements.

When the difference in energy consumption between the sensors is not significant, then the algorithm works like the hysteretic Q-learning algorithm. Moreover, if there are too many sensors, the weight of the energy consumption of a single sensor become insignificant compared to the overall consumption of all sensors. A possible solution may be to cluster the sensors with similar energy consumptions in the same agent.

Conclusion

The energy management of an embedded system is a difficult problem. Indeed, a sensor node is composed of different modules, which impacts the energy consumption of the overall node. The use of a single agent limits the size of the MDP and makes it difficult to control each module independently. A MARL approach allows the independent control of the different modules and to separate the optimization of the frequency measurements and the optimization of the communication. Instead of immediately transmitting data, we control the transmission depending on the load of a data buffer with a specific agent. This is a more energy efficient approach. And it is easier to improve a MAS approach since we can modify only one agent to increase the energy efficiency of the overall sensor node.

Moreover, a sensor node often has several sensors with different energy consumptions. As the proposed approach controls independently the different sensors on the node, we can reduce the measurement frequency of the most consuming sensors in a first step, allowing us to improve the QoS of the system. Furthermore, the data buffer fills up slower, which also reduces the energy consumption of the communication module.

In addition, the proposed approach reduces the memory requirements in exchange of a little increase of the computation in comparison to a single agent approach. A system with two sensing agents reduces the memory requirement by 80%, whereas the computation consists on updating two Q-values instead of one.

Conclusions and Perspectives

Conclusions

Energy management is one of the main challenges in the design of wireless sensor nodes, especially for applications with a long life-span. Indeed, typical wireless sensor nodes are battery-powered and batteries can only store a finite amount of energy. A solution is to enable each node to harvest energy directly from its environment. As such energy sources are dynamic and uncontrolled, it is required to perform on-line adaptation of the nodes performance in order to maximize the application performance, while avoiding power failures. As shown in this PhD thesis, reinforcement learning is a very promising approach to the energy management of a sensor node. Indeed, reinforcement learning approaches allow sensor nodes to learn how to take actions under uncertainties in energy sources. It promotes good decisions to adapt the node's behaviour to the available energy.

In the first chapter, we provided an introduction on reinforcement learning and presented the theory behind it. We also introduced the neural networks, explained how they work since they are used in one of the reinforcement algorithm used in this thesis, namely deep Q-learning.

In the second chapter, we have seen that the reinforcement learning approach provides a powerful tool for the optimization of the energy consumption of embedded systems and that it has gained a certain popularity in the last years. Nevertheless, several challenges must be addressed before using it in an application. The first one is the selection of the appropriate algorithm, which is not trivial. Indeed, there is a lack of metric to help a designer to select the reinforcement learning algorithm depending on a given application. Then, the designer needs to configure many parameters: the leaning rate, the discount factor, the reward function. The configuration is typically done empirically or using the expertise of the designer. The reward function determines the behaviour of the node. However, the definition of the reward function is often not presented in the literature. In this thesis, we have compared different reinforcement learning approaches and proposed different metrics to help a designer to select the appropriate one for its application. We also compared different reward functions to determine a way to construct them, then we

proposed two distinct reward functions which adjust the performance according to the battery charge level. There is another difficulty, i.e., the use of a look-up table in Q-learning limits the size of the decision process and makes it less scalable for more different states and actions. To overcome this, we proposed an energy management approach of a sensor node based on Multi-Agent Reinforcement Learning where each hardware module is controlled independently by a learning agent with its own decision process.

In the third chapter, we explored the use of different reinforcement learning approaches for the energy management of a sensor node. We presented the Q-learning algorithm which uses a look-up table to store the knowledge of the agent. Several variants exist and we presented two of them: Dyna Q-learning and Deep Q-learning. Dyna Q-learning also uses a look-up table to store the knowledge, but in addition it builds a model of the dynamics of the environment to accelerate the learning convergence. Deep Q-learning is a neural network version of Q-learning. Instead of storing the learning in a look-up table, at each step, it computes the expected reward for each possible actions and uses the feedback of the reward to optimize the weights of connections between neurons. We compared the different versions for the energy management of a marine buoy use case. We proposed different metrics to select the appropriate approach depending on the capabilities of the system and the application constraints. We have shown that Q-learning is the less processing hungry approach, but requires more memory. However, it is the most stable algorithm, only one value is updated in the look-up table at each action. The Deep Q-learning algorithm is much more processing hungry since for each decision, it computes the reward for all the possible actions. Dyna Q-learning requires more memory than the two other algorithms, but the learning speed is accelerated.

Determining the appropriate approach to use for a given application is not sufficient. The designer also needs to design a reward function which promotes a behaviour to balance the performance and the energy consumption in order to keep the sensor node alive. In this way, in the fourth chapter, we evaluated and compared different reward functions to identify the parameters to use in order to design an efficient reward function. We found that the use of a balancing parameter between the performance and the battery charge level is an efficient approach allowing the designer to give more weight for a performance criterion or to save battery power. Then, in a second part, we proposed two reward functions able to balance the performance and the battery charge level. The objective was to maximize the performance when the battery is full and to preserve it when it discharges. The first one adapts the balance between the performance and the battery

level at different level of battery charge. Several parameters must be set i.e., the number of level, the importance of the different criterion at each level, the limit of the level. This reward function is parametrizable, but requires more work and expertise from the designer. This is why we proposed a generalisation of the first proposed reward function, i.e., we use the battery charge level to balance the performance and the energy consumption. There is no parameter to tune which makes it easier to apply. It also accelerates the learning speed compared to the previous reward function.

The single agent approach is limited by the size of the decision process. To have a finer control on the energy consumption of the system, the agent must have multiple possible actions but the addition of more actions to the decision process increases its size. A solution is to divide the complex problem into smaller ones that are easier to solve. In the last chapter, we proposed to consider each hardware module as an agent and to apply a multi-agent reinforcement learning approach. Moreover, nodes have several sensors with different energy consumptions. So, we proposed an algorithm which adjusts the measurement frequency of the sensors according to their respective energy consumption. The proposed algorithm does not need to know the energy consumption of the sensor, it computes a coefficient which correspond to the variation of the battery charge level when the sensor works alone over the sum of the variations for all the sensors. We applied a multi-agent reinforcement learning algorithm, which adjusts the penalty when the system reaction to a joint action is detrimental. Thus, a high consuming sensor has its frequency measurements reduced sooner than a sensor which consumes less energy which preserves the QoS of the node. The proposed approach is applied to a marine buoy use case with two different sensors; it reduces the memory requirement by 80%, whereas the computation consists of updating two Q-values instead of one.

In this thesis, we explored the use of reinforcement learning approaches for the energy management of a sensor node. We focused our work on a popular algorithm, Q-learning, and different variants of it. The selection of the appropriate algorithm for a given application depends on the memory and processing capabilities available on the platform used. Moreover, in simulations the learning could takes several days to complete with a relatively small decision process. Thus, these approaches are not well suited if the system must be efficient quickly after its deployment or rapidly adapt to a new evolution of the environment. We also observe that a good reward function could improve the learning speed. The use of multi agent reinforcement learning does not accelerate the learning speed but it allows to use look up table to solve more complex decision process to ease

their use on real systems.

Perspectives

This section presents perspectives opened by our work and, in our opinion, are worthwhile subjects for future works. In this section, we present three works from the shortest to the longest term.

Application to a real system The first perspective would be the application of the work done in simulation during the thesis on a real system. Q-learning is a popular reinforcement learning algorithm widely used in the literature, we used it in a simulation environment to manage the energy consumption of a sensor node. However, we would have liked to test it on a real marine buoy to validate the behaviour observed in simulation. Indeed, it is challenging to model completely the complexity of the environment and the system such as the ageing of the components, especially the solar panel whose performance decreases significantly in a marine environment due to the layers of salts that will cover it. The approaches developed during the thesis can be implemented quite quickly, the most time-consuming part would be the actual deployment of the buoy at sea.

Multi-Objective Reinforcement Learning A second perspective would be the use of multi-objective approaches to design reward functions able to balance more than two different objectives. Many real-world problems have multiple, possibly conflicting, objectives and the majority of reinforcement learning research and applications still assume only a single objective. Thus, in this thesis, we proposed two scalar reward functions which balance a performance criterion and the energy consumption. However, there are often more than two criteria to balance and the design of efficient reward function, in this case, is difficult. For instance, in a network routing the criteria may consist of energy consumption, latency, and channel capacity, which are conflicting objectives. That is why, recently, there has been growing interest in solving Multi-Objective Reinforcement Learning (MORL), where the notion of optimality is replaced by Pareto optimality, a concept for representing compromises among the objectives. A future work may include this approach to reward several tasks, i.e., maximizing the battery charge level of the node and the QoS, and minimizing the channel use.

Transfer learning A third perspective would be to accelerate the learning speed to make the use of reinforcement learning approach interesting for more challenging applications where the system needs to be operational fast. An interesting work is the transfer learning. The main idea of transfer learning is that experience gained in learning to perform one task can help improve learning performance in a related, but different, task or in new conditions. In this way, an agent can learn how to act and then, transfer the knowledge among different sensor nodes. For instance, a marine drone can be trained to follow the border of a polluted area in a simulator where different environmental parameters vary to learn a more flexible policy. We can add noise in the environment to increase realism to avoid training a policy that exploits a physically implausible phenomenon of the simulator. Another advantage of transfer learning is the possibility for e.g. a drone to share its experience with a newly arrived drone in the swarm. This possibility allows to speed up the learning speed while making the policy more efficient in the real environment.

References

[1] Sam Lucero et al. *IoT Platforms : Enabling the Internet of Things*. IHS markit, White paper, 2016.

[2] Gyuhae Park, Tajana Rosing, Michael D Todd et al. « Energy harvesting for structural health monitoring sensor networks ». In : *Journal of Infrastructure Systems* 14.1 (2008), p. 64-79.

[3] Sennur Ulukus, Aylin Yener, Elza Erkip et al. « Energy harvesting wireless communications : A review of recent advances ». In : *IEEE Journal on Selected Areas in Communications* 33.3 (2015), p. 360-381.

[4] Faisal Karim Shaikh et Sherali Zeadally. « Energy harvesting in wireless sensor networks : A comprehensive review ». In : *Renewable and Sustainable Energy Reviews* 55 (2016), p. 1041-1054.

[5] Deepak Mishra, Swades De, Soumya Jana et al. « Smart RF energy harvesting communications : Challenges and opportunities ». In : *IEEE Communications Magazine* 53.4 (2015), p. 70-78.

[6] Vikrant Sharma et SS Chandel. « Performance and degradation analysis for long term reliability of solar photovoltaic systems : a review ». In : *Renewable and Sustainable Energy Reviews* 27 (2013), p. 753-767.

[7] Albert Crespo-Yepes, E Barajas, Javier Martin-Martinez et al. « MOSFET degradation dependence on input signal power in a RF power amplifier ». In : *Microelectronic Engineering* 178 (2017), p. 289-292.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver et al. « Playing atari with deep reinforcement learning ». In : *arXiv preprint arXiv :1312.5602* (2013).

[9] Jens Kober, J Andrew Bagnell et Jan Peters. « Reinforcement learning in robotics : A survey ». In : *The International Journal of Robotics Research* 32.11 (2013), p. 1238-1274.

[10] LA Prashanth et Shalabh Bhatnagar. « Reinforcement learning with function approximation for traffic signal control ». In : *IEEE Transactions on Intelligent Transportation Systems* 12.2 (2011), p. 412-421.

[11] Rui Xiong, Jiayi Cao et Quanqing Yu. « Reinforcement learning-based real-time power management for hybrid energy storage system in the plug-in hybrid electric vehicle ». In : *Applied energy* 211 (2018), p. 538-548.

[12] R S Sutton et A G Barto. « Reinforcement learning : An introduction ». In : *Neural Networks IEEE Transactions on* 9 (2013), p. 1054.

[13] Richard E. Bellman. *Dynamic Programming*. 1957.

[14] Richard E Bellman. *Adaptive control processes : a guided tour*. T. 2045. Princeton university press, 2015.

[15] Shaswot Shresthamali, Masaaki Kondo et Hiroshi Nakamura. « Adaptive Power Management in Solar Energy Harvesting Sensor Node Using Reinforcement Learning ». In : *ACM Transactions on Embedded Computing Systems* 16.5s (2017), p. 1-21. issn : 15399087. doi : 10.1145/3126495.

[16] Liu Zhenzhen, Elhanany Itamar, Zhenzhen Liu et al. « RL-MAC : A QoS-Aware Reinforcement Learning based MAC Protocol for Wireless Sensor Networks ». In : *Icnsc'06* (2006), p. 768-773. doi : 10.1109/ICNSC.2006.1673243.

[17] Jeremy Gummeson, Deepak Ganesan, Mark D Corner et al. « An adaptive link layer for heterogeneous multi-radio mobile sensor networks ». In : *IEEE Journal on Selected Areas in Communications* 28.7 (2010), p. 1094-1104.

[18] Michele Chincoli et Antonio Liotta. « Self-learning power control in wireless sensor networks ». In : *Sensors* 18.2 (2018), p. 375.

[19] Debashis Banerjee, Shreyas Sen et Abhijit Chatterjee. « Self learning analog/mixed-signal/RF systems : Dynamic adaptation to workload and environmental uncertainties ». In : *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press. 2015, p. 59-64.

[20] Xuedong Liang, Min Chen, Yang Xiao et al. « MRL-CC : a novel cooperative communication protocol for QoS provisioning in wireless sensor networks ». In : ().

[21] Hongbin Chen, Xueyan Li et Feng Zhao. « A reinforcement learning-based sleep scheduling algorithm for desired area coverage in solar-powered wireless sensor networks ». In : *IEEE Sensors Journal* 16.8 (2016), p. 2763-2774.

[22] Richard Bellman. « On a routing problem ». In : *Quarterly of applied mathematics* 16.1 (1958), p. 87-90.

[23] Yishan Su, Rong Fan, Xiaomei Fu et al. « DQELR : An Adaptive Deep Q-Network-Based Energy-and Latency-Aware Routing Protocol Design for Underwater Acoustic Sensor Networks ». In : *IEEE Access* 7 (2019), p. 9091-9104.

[24] Hu Tan, Lijun Zhao, Wei Liu et al. « Adaptive congestion avoidance scheme based on reinforcement learning for wireless sensor network ». In : (2011).

[25] Muhidul Islam Khan, Kewen Xia, Ahmad Ali et al. « Energy-aware task scheduling by a true online reinforcement learning in wireless sensor networks ». In : *International Journal of Sensor Networks* 25.4 (2017), p. 244-258.

[26] Yvan Debizet, Guenole Lallement, Fady Abouzeid et al. « Q-Learning-based Adaptive Power Management for loT System-an-Chips with Embedded Power States ». In : *2018 IEEE International Symposium on Circuits and Systems (ISCAS)* (2018).

[27] Christopher J. C. H. Watkins et Peter Dayan. « Q-learning ». In : *Machine Learning* 8.3-4 (mai 1992), p. 279-292. URL : http://link.springer.com/10.1007/BF00992698.

[28] Hao Shen, Ying Tan, Jun Lu et al. « Achieving Autonomous Power Management using Reinforcement Learning ». In : *ACM Transactions on Design Automation of Electronic Systems* 18.2 (2013), p. 1-32. ISSN : 10844309. DOI : 10.1145/2442087.2442095.

[29] A Molnos, S Lesecq, J Mottin et al. « Investigation of Q-learning applied to DVFS management of a System-on-Chip ». In : *IFAC-PapersOnLine* 49 (2016), p. 278-284. URL : www.sciencedirect.com.

[30] Salvatore Carta, Andrea Alimonda, Alessandro Pisano et al. « A control theoretic approach to energy-efficient pipelined computation in MPSoCs ». In : *ACM Transactions on Embedded Computing Systems (TECS)* 6.4 (2007), p. 27.

[31] Andrea Alimonda, Salvatore Carta, Andrea Acquaviva et al. « A feedback-based approach to dvfs in data-flow applications ». In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28.11 (2009), p. 1691-1704.

[32] Wendi B Heinzelman, Anantha P Chandrakasan, Hari Balakrishnan et al. « An application-specific protocol architecture for wireless microsensor networks ». In : *IEEE Transactions on wireless communications* 1.4 (2002), p. 660-670.

[33] Justin A Boyan et Michael L Littman. « Packet routing in dynamically changing networks : A reinforcement learning approach ». In : *Advances in neural information processing systems*. 1994, p. 671-678.

[34] Lester Randolph Ford Jr. et Delbert Ray Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[35] Tiansi Hu et Yunsi Fei. « QELAR : A machine-learning-based adaptive routing protocol for energy-efficient and lifetime-extended underwater sensor networks ». In : *IEEE Transactions on Mobile Computing* 9.6 (2010), p. 796-809.

[36] Peng Xie, Jun-Hong Cui et Li Lao. « VBF : Vector-Based Forwarding Protocol for Underwater Sensor Networks ». In : *NETWORKING 2006. Networking Technologies, Services, and Protocols ; Performance of Computer and Communication Networks ; Mobile and Wireless Communications Systems*. Sous la dir. de Fernando Boavida, Thomas Plagemann, Burkhard Stiller et al. Berlin, Heidelberg : Springer Berlin Heidelberg, 2006, p. 1216-1221. isbn : 978-3-540-34193-2.

[37] Wenjing Guo, Cairong Yan et Ting Lu. « Optimizing the lifetime of wireless sensor networks via reinforcement-learning-based routing ». In : *International Journal of Distributed Sensor Networks* 15.2 (2019). doi : 10.1177/1550147719833541.

[38] Rahul C Shah et Jan M Rabaey. « Energy aware routing for low energy ad hoc sensor networks ». In : *2002 IEEE Wireless Communications and Networking Conference Record. WCNC 2002 (Cat. No. 02TH8609)*. T. 1. IEEE. 2002, p. 350-355.

[39] Samira Yessad, Nassima Tazarart, Lyes Bakli et al. « Balanced energy efficient routing protocol for WSN ». In : *2012 International Conference on Communications and Information Technology (ICCIT)*. IEEE. 2012, p. 326-330.

[40] A Pravin Renold et S Chandrakala. « MRL-SCSO : multi-agent reinforcement learning-based self-configuration and self-optimization protocol for unattended wireless sensor networks ». In : *Wireless Personal Communications* 96.4 (2017), p. 5061-5079.

[41] Wei Ye, John Heidemann et Deborah Estrin. « An energy-efficient MAC protocol for wireless sensor networks ». In : *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*. T. 3. IEEE. 2002, p. 1567-1576.

[42] Lei Zhou, Hao Tang, Huizi Li et al. « Dynamic power management strategies for a sensor node optimised by reinforcement learning ». In : *International Journal of Computational Science and Engineering* 13.1 (2016), p. 24-37. doi : 10.1504/IJCSE.2016.10000008.

[43] Selahattin Kosunalp. « A new energy prediction algorithm for energy-harvesting wireless sensor networks with Q-Learning ». In : *IEEE Access* 4 (2016), p. 5755-5763.

[44] Aman Kansal, Jason Hsu, Sadaf Zahedi et al. « Power management in energy harvesting sensor networks ». In : *ACM Transactions on Embedded Computing Systems (TECS)* 6.4 (2007), p. 32.

[45] Dong Kun Noh et Kyungtae Kang. « Balanced energy allocation scheme for a solar-powered sensor system and its effects on network-wide performance ». In : *Journal of Computer and System Sciences* 77.5 (2011), p. 917-932.

[46] Alessandro Cammarano, Chiara Petrioli et Dora Spenza. « Pro-Energy : A novel energy prediction model for solar and wind energy-harvesting wireless sensor networks ». In : *2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012)*. IEEE. 2012, p. 75-83.

[47] Yanzhi Wang, Qing Xie, Ahmed Ammari et al. « Deriving a Near-optimal Power Management Policy Using Model-free Reinforcement Learning and Bayesian Classification ». In : *Proceedings of the 48th Design Automation Conference*. San Diego, California : ACM, 2011, p. 41-46. isbn : 978-1-4503-0636-2. doi : 10.1145/2024724.2024735.

[48] Chi-Hong Hwang et Allen C-H Wu. « A predictive system shutdown method for energy saving of event-driven computation ». In : *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 5.2 (2000), p. 226-241.

[49] Richard S Sutton, Andrew G Barto et Ronald J Williams. « Reinforcement learning is direct adaptive optimal control ». In : *IEEE Control Systems Magazine* 12.2 (1992), p. 19-22.

[50] Muhidul Islam Khan et Kewen Xia. « Energy-Aware Task Scheduling by a True Online Reinforcement Learning in Wireless Sensor Networks ». In : *Article in International Journal of Sensor Networks* (2016). doi : 10.1504/IJSNET.2016.10001403.

[51] Kunal Shah et Mohan Kumar. « Distributed independent reinforcement learning (DIRL) approach to resource management in wireless sensor networks ». In : *2007 IEEE International Conference on Mobile Adhoc and Sensor Systems*. IEEE. 2007, p. 1-9.

[52] Muhidul Islam Khan et Bernhard Rinner. « Resource coordination in wireless sensor networks by cooperative reinforcement learning ». In : *2012 IEEE International Conference on Pervasive Computing and Communications Workshops*. IEEE. 2012, p. 895-900.

[53] Muhidul Islam Khan et Bernhard Rinner. « Energy-aware task scheduling in wireless sensor networks based on cooperative reinforcement learning ». In : *2014 IEEE International Conference on Communications Workshops (ICC)*. IEEE. 2014, p. 871-877.

[54] Muhidul Islam Khan. « Resource-aware task scheduling by an adversarial bandit solver method in wireless sensor networks ». In : *EURASIP Journal on Wireless Communications and Networking* 2016.1 (2016), p. 10.

[55] David Silver, Julian Schrittwieser, Karen Simonyan et al. « Mastering the game of go without human knowledge ». In : *Nature* 550.7676 (2017), p. 354.

[56] Qingchen Zhang, Man Lin, Laurence T Yang et al. « Energy-efficient scheduling for real-time systems based on deep Q-learning model ». In : *IEEE Transactions on Sustainable Computing* (2017).

[57] Fakhruddin Muhammad Mahbub ul Islam et Man Lin. « Hybrid DVFS scheduling for real-time systems based on reinforcement learning ». In : *IEEE Systems Journal* 11.2 (2015), p. 931-940.

[58] Minghui Min, Liang Xiao, Ye Chen et al. « Learning-based computation offloading for IoT devices with energy harvesting ». In : *IEEE Transactions on Vehicular Technology* 68.2 (2019), p. 1930-1941.

[59] Liang Xiao, Yanda Li, Xueli Huang et al. « Cloud-based malware detection game for mobile devices with offloading ». In : *IEEE Transactions on Mobile Computing* 16.10 (2017), p. 2742-2750.

[60] Ji Li, Hui Gao, Tiejun Lv et al. « Deep reinforcement learning based computation offloading and resource allocation for MEC ». In : *2018 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2018, p. 1-6.

[61] Peng Xie, Jun-Hong Cui et Li Lao. « VBF : vector-based forwarding protocol for underwater sensor networks ». In : *International conference on research in networking*. Springer. 2006, p. 1216-1221.

[62] Guihong Chen, Yiju Zhan, Geyi Sheng et al. « Reinforcement Learning-Based Sensor Access Control for WBANs ». In : *IEEE Access* 7 (2019), p. 8483-8494. ISSN : 2169-3536. DOI : 10.1109/ACCESS.2018.2889879.

[63] Seungku Kim et Doo-Seop Eom. « Link-state-estimation-based transmission power control in wireless body area networks ». In : *IEEE journal of Biomedical and Health Informatics* 18.4 (2013), p. 1294-1302.

[64] Guihong Chen, Yiju Zhan, Ye Chen et al. « Reinforcement learning based power control for in-body sensors in WBANs against jamming ». In : *IEEE Access* 6 (2018), p. 37403-37412.

[65] Fayçal Ait Aoudia, Matthieu Gautier et Olivier Berder. « Learning to Survive : Achieving Energy Neutrality in Wireless Sensor Networks Using Reinforcement Learning ». In : (2017).

[66] Olivier Berder et Olivier Sentieys. « Powwow : Power optimized hardware/software framework for wireless motes ». In : *23th International Conference on Architecture of Computing Systems 2010*. VDE. 2010, p. 1-5.

[67] S Peng et CP Low. « Prediction free energy neutral power management for energy harvesting wireless sensor nodes ». In : *Ad Hoc Networks* 13 (2014), p. 351-367.

[68] Fayçal Ait Aoudia, Matthieu Gautier et Olivier Berder. « Fuzzy power management for energy harvesting Wireless Sensor Nodes ». In : *2016 IEEE International Conference on Communications (ICC)*. IEEE. 2016, p. 1-6.

[69] Christopher M Vigorito, Deepak Ganesan et Andrew G Barto. « Adaptive control of duty cycling in energy-harvesting wireless sensor networks ». In : *2007 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*. IEEE. 2007, p. 21-30.

[70] Muhidul Islam Khan, Muhammad Mahtab Alam et Yannick Le Moullec. « A multi-armed bandit solver method for adaptive power allocation in device-to-device communication ». In : *Procedia computer science* 130.C (2018), p. 1069-1076.

[71] A Meindl. « Guide to Moored Buoys and Other Ocean Data Acquisition Systems. » In : (1996).

[72] Reinaldo AC Bianchi, Carlos HC Ribeiro et Anna HR Costa. « Heuristically Accelerated Q–Learning : a new approach to speed up Reinforcement Learning ». In : *Brazilian Symposium on Artificial Intelligence*. Springer. 2004, p. 245-254.

[73] Richard S Sutton. « Dyna, an integrated architecture for learning, planning, and reacting ». In : *ACM Sigart Bulletin* 2.4 (1991), p. 160-163.

[74] Volodymyr Mnih, Koray Kavukcuoglu, David Silver et al. « Human-level control through deep reinforcement learning ». In : *Nature* 518.7540 (2015), p. 529.

[75] David Silver, Aja Huang, Chris J Maddison et al. « Mastering the game of Go with deep neural networks and tree search ». In : *nature* 529.7587 (2016), p. 484.

[76] Ruishan Liu et James Zou. « The Effects of Memory Replay in Reinforcement Learning ». In : *CoRR* abs/1710.06574 (2017). arXiv : 1710.06574. URL : http://arxiv.org/abs/1710.06574.

[77] Vinod Nair et Geoffrey E Hinton. « Rectified linear units improve restricted boltzmann machines ». In : *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, p. 807-814.

[78] Hado Van Hasselt, Arthur Guez et David Silver. « Deep reinforcement learning with double q-learning ». In : *Thirtieth AAAI conference on artificial intelligence*. 2016.

[79] Tom Schaul, John Quan, Ioannis Antonoglou et al. « Prioritized experience replay ». In : *arXiv preprint arXiv :1511.05952* (2015).

[80] Matteo Hessel, Joseph Modayil, Hado Van Hasselt et al. « Rainbow : Combining improvements in deep reinforcement learning ». In : *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[81] Maria GORLATOVA, John SARIK, Guy GREBLA et al. « Movers and shakers : Kinetic energy harvesting for the internet of things ». In : *ACM SIGMETRICS Performance Evaluation Review*. T. 42. 1. ACM. 2014, p. 407-419.

[82] MA WIERING. « Multi-agent reinforcement learning for traffic light control ». In : *Machine Learning : Proceedings of the Seventeenth International Conference (ICML'2000)*. 2000, p. 1151-1158.

[83] Monireh ABDOOS, Nasser MOZAYANI et Ana LC BAZZAN. « Traffic light control in non-stationary environments based on multi agent Q-learning ». In : *2011 14th International IEEE conference on intelligent transportation systems (ITSC)*. IEEE. 2011, p. 1580-1585.

[84] Carolina HIGUERA, Fernando LOZANO, Edgar Camilo CAMACHO et al. « Multiagent Reinforcement Learning Applied to Traffic Light Signal Control ». In : *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Springer. 2019, p. 115-126.

[85] Martin LAUER et Martin RIEDMILLER. « An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems ». In : *Proceedings of the Seventeenth International Conference on Machine Learning*. Morgan Kaufmann, 2000, p. 535-542.

[86] Laëtitia MATIGNON, Guillaume J. LAURENT et Nadine LE FORT-PIAT. « Hysteretic Q-Learning : an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. » In : *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'07*. San Diego, CA., United States, oct. 2007, p. 64-69. URL : https://hal.archives-ouvertes.fr/hal-00187279.

Abbreviations

CPS	**C**yber-**P**hysical **S**ystem
IoT	**I**nternet-**o**f-**T**hings
OS	**O**perating **S**ystem
RL	**R**einforcement **L**earning
MDP	**M**arkov **D**ecision **P**rocess
DP	**D**ynamic **P**rogramming
ANN	**A**rtificial **N**eural **N**etwork
QoS	**Q**uality **o**f **S**ervice
WBAN	**W**ireless **B**ody **A**rea **N**etwork
BLE	**B**luetooth **L**ow **E**nergy
MCU	**M**icro-**C**ontroller **U**nit
MARL	**M**ulti-**A**gent **R**einforcement **L**earning
MAS	**M**ulti-**A**gent **S**ystem
SG	**S**tochastic **G**ame
MMDP	**M**ulti-agent **M**arkov **D**ecision **P**rocess

Summary of Notation

γ	Discount factor
α	Learning rate
$\mathcal{A}$	Action space
$\mathcal{S}$	State space
$\mathcal{R}$	Reward function
$\mathcal{T}$	State transition probability density function
π	Policy in reinforcement learning theory
π^*	Optimal policy in reinforcement learning theory
Q^*	Optimal state-action value function in reinforcement learning
Q^π	State-action value function in reinforcement learning
V^*	Optimal state value function in reinforcement learning
V^π	State value function in reinforcement learning
a_t	Action taken at the step t
s_t	State at the step t
r_t	Reward received at the step t
λ	Decay parameter of the algorithms with eligibility traces

List of Figures

1 Examples of different cyber-physical system applications 6
2 IoT connected devices installed base worldwide from 2015 to 2025 (in billions) (source IHS [1]) 7
3 Schema of an IoT node 8
4 Energy consuming tasks of a node 9
5 The design of an energy model for an IoT node faces many hurdles 10

1.1 Interaction agent-environment 16
1.2 Model of a Artificial Neuron 22
1.3 Structure example of a multilayer perceptron with a single hidden layer . . 23

2.1 Markov Decision Process diagram showing the set of actions and transitions [26] 29
2.2 Frame structure employed by the RL-MAC protocol [16] 36
2.3 Reward function [15] 42

3.1 Energy harvested in Lorient, France, by the buoy during a deployment of 21 days 55
3.2 Marine buoy 56
3.3 Evolution of the battery charge level and sampling frequency of the sensors using the Q-learning algorithm 58
3.4 Evolution of the battery charge and sampling frequency of the sensors using the Dyna Q-learning algorithm 61
3.5 Deep Q-learning 62
3.6 Activation function: linear rectifier (Eq. 3.5) (ReLU [77]) 63
3.7 Evolution of the battery charge and sampling frequency of the sensors using the Deep Q-learning algorithm 64
3.8 Algorithms comparison in terms of memory and computational requirements, learning speed and stability 66

4.1 Sensor node fitted on a chest to monitor the heart beat 71

4.2 Normalised average energy consumption of the node according to 1) the activity of the wearer and 2) the reward function used within the Q-learning algorithm. R1, R2 and R5 behave as expected since they allow the node to consume more when more energy is harvested. 75
4.3 Evolution of the battery charge level and frequency measurements of sensors using the Q-learning algorithm with the reward function of Equation 4.1 . 78
4.4 Evolution of the battery charge level and measurements frequency of sensors using the Q-learning algorithm with the reward function 4.1 79
4.5 Evolution of the battery charge level and measurements frequency of sensors using the Q-learning algorithm with the reward function in Eq. 4.2 81
4.6 Evolution of the battery charge level and measurements frequency of sensors using the Q-learning algorithm with the reward function in Eq. 4.2 82
4.7 Evolution of the battery charge level and measurements frequency of sensors using the Q-learning algorithm with the reward function 4.2 83

5.1 Multi-Agent Reinforcement Learning in which three agents share the same environment. Each agent makes an action and the combination of all actions has an impact on the state of the environment, and each agent receives a reward depending on the state of the environment. 89
5.2 Distributed Energy Management of a wireless sensor node. Two sensing agents control the measurement frequency of the anemometer sensor and the atmospheric sensor, respectively, according to the battery charge level. The measured data fill a buffer, and a communication agent controls the data transmission according to the filling of the data buffer. 92
5.3 Q-table of the agent for the atmospheric sensor 95
5.4 Q-table of the agent for the atmospheric sensor for battery charge level between 100% and 60% 96
5.5 Q-table of the agent for the atmospheric sensor for battery charge level between 60% and 20% 96
5.6 Q-table of the agent for the atmospheric sensor for battery charge level between 20% and 0% 97
5.7 Q-table of agent for the anemometer 97
5.8 Q-table of agent for the anemometer for battery charge level between 100% and 70% 98

5.9 Q-table of agent for the anemometer for battery charge level between 70% and 30% 98
5.10 Q-table of agent for the anemometer for battery charge level between 30% and 0% 98

List of Tables

1 Power density of energy harvesting technologies 8

2.1 Performance comparison of RLBR [37] and state-of-the-art according to the different definitions of a network lifetime 35
2.2 Different operating mode of the sensor node [42] 38
2.3 Performance comparison of QL-SEP with state-of-the-art 40

3.1 Buoy components 56

4.1 Node components and respective current consumptions 71
4.2 Kinetic motion's power harvested for three different activities 72
4.3 Set of actions with both different processor frequencies (F_p), periods between each measurement (P_s), and the associated average current consumption 72
4.4 Evaluation of the different reward functions 76

Personal Publications

International Journals

- Yohann Rioual, Johann Laurent, Jean-Philippe Diguet. Reinforcement Learning Approaches Guidelines for Energy Management Algorithm. Journal of Low Power Electronics, American Scientific Publishers, Sep 2019

International Conferences

- Yohann Rioual, Johann Laurent, Eric Senn, Jean-Philippe Diguet. Reinforcement Learning Strategies for Energy Management in Low Power IoT. CSCI, Dec 2017, Las Vegas, United States.
- Yohann Rioual, Yannick Le Moullec, Johann Laurent, Muhidul Islam Khan, Jean-Philippe Diguet. Reward Function Evaluation in a Reinforcement Learning Approach for Energy Management. 2018 16th Biennial Baltic Electronics Conference (BEC), Oct 2018, Tallinn, Estonia. pp.1-4.

National Conferences

- Yohann Rioual. Environnement de simulation pour IoT faible consommation. CNRS GdR SoC2, Journée thématique "Near Sensor Computing", Nov 2017, Paris, France.
- Yohann Rioual, Jean-Philippe Diguet, Johann Laurent, Eric Senn. Simulation Approach for Energy Management in Wireless Sensor Networks. BEE Week 2017, Nov 2017, Bordeaux, France.

Titre : Gestion de l'Energie basée sur l'Apprentissage par Renforcement pour les Systèmes Cyber-Physiques Autonomes

Mots clés : Apprentissage par Renforcement, gestion d'énergie, système cyber-physique

Résumé : La gestion d'énergie d'un système cyber physique est une tâche difficile à cause de la complexité des architectures matérielles et l'utilisation d'OS. En outre, ces systèmes sont déployés dans des environnements qui évoluent et où leur capacité de recharge en énergie varie. Avec le temps, leur consommation en énergie est modifiée du fait du vieillissement des composants. Les modèles de consommation conçus en laboratoire ne peuvent pas prendre en compte tous les scénarios de déploiement possible ainsi que le vieillissement du système. Une approche qui se développe est l'utilisation d'apprentissage par renforcement dans lequel nous n'avons plus connaissance du modèle de consommation du système ; mais grâce à cette approche, ce dernier est capable d'adapter son fonctionnement pendant son déploiement en fonction de l'évolution de son environnement.

Plusieurs approches existent en apprentissage par renforcement. La première partie de cette thèse est consacrée à la proposition de lignes directrices pour aider à la sélection de l'approche la plus appropriée pour une application et une cible donnée.

La deuxième partie se concentre sur la conception de la récompense que l'on donne à notre système et qui va influencer son comportement dans son environnement. Deux fonctions de récompense capables d'ajuster la performance du système en fonction de l'énergie disponible y sont présentées.

La troisième et dernière partie explore l'utilisation de plusieurs agents pour controler independamment les différents modules de notre système. Cette approche permet un contrôle plus précis de la consommation en énergie, réduisant l'utilisation de mémoire par rapport à une approche avec un agent unique.

Title : RL-based Energy Management for Autonomous Cyber Physical Systems

Keywords : Reinforcement Learning, energy management, cyber physical system

Abstract: The energy management of a cyber physical system is a difficult task because of the complexity of hardware architectures and the use of OS. In addition, these systems are deployed in changing environments where their energy harvesting capacity varies. Over time, their energy consumption is modified due to the ageing of the components. Consumption models designed in the laboratory cannot take into account all possible deployment scenarios and system aging. One approach that is developing is the use of reinforcement learning in which we no longer know the system's consumption model; but thanks to this approach, the system is still able to adapt its operation during its deployment according to the evolution of its environment.

Several approaches exist in reinforcement learning. The first part of this thesis is devoted to proposing guidelines to help for selecting the most appropriate approach for a given application and target.

The second part of this thesis focuses on the design of the reward we give to our system that will influence its behaviour in its environment. Two reward functions able to adjust the system's performance according to the energy available are presented.

The third and last part of this thesis explores the use of several agents to independently control the different modules of our system. This approach allows a more precise control of energy consumption, reducing memory usage compared to a single agent approach.