Received August 5, 2019, accepted August 24, 2019, date of publication August 29, 2019, date of current version September 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2938390

A Reinforcement Learning-Based QAM/PSK Symbol Synchronizer

MARCO MATTA[1], GIAN CARLO CARDARILLI[1], (Member, IEEE), LUCA DI NUNZIO[1], ROCCO FAZZOLARI[1], DANIELE GIARDINO[1], ALBERTO NANNARELLI[2], (Senior Member, IEEE), MARCO RE[1], (Member, IEEE), AND SERGIO SPANÒ[1]

[1]Department of Electronic Engineering, University of Rome Tor Vergata, 00133 Rome, Italy
[2]Department of Applied Mathematics and Computer Science, Danmarks Tekniske Universitet, 2800 Kgs. Lyngby, Denmark

Corresponding author: Marco Matta (matta@ing.uniroma2.it)

ABSTRACT Machine Learning (ML) based on supervised and unsupervised learning models has been recently applied in the telecommunication field. However, such techniques rely on application-specific large datasets and the performance deteriorates if the statistics of the inference data changes over time. Reinforcement Learning (RL) is a solution to these issues because it is able to adapt its behavior to the changing statistics of the input data. In this work, we propose the design of an RL Agent able to learn the behavior of a Timing Recovery Loop (TRL) through the Q-Learning algorithm. The Agent is compatible with popular PSK and QAM formats. We validated the RL synchronizer by comparing it to the Mueller and Müller TRL in terms of Modulation Error Ratio (MER) in a noisy channel scenario. The results show a good trade-off in terms of MER performance. The RL based synchronizer loses less than 1 dB of MER with respect to the conventional one but it is able to adapt its behavior to different modulation formats without the need of any tuning for the system parameters.

INDEX TERMS Artificial intelligence, machine learning, reinforcement learning, Q-learning, synchronization, timing recovery loop.

I. INTRODUCTION

Machine Learning (ML) is a field of Artificial Intelligence based on statistical methods to enhance the performance of algorithms in data pattern identification [1]. ML is applied in several fields, such as medicine [2], financial trading [3], big data management [4], imaging and image processing [5], security [6], [7], mobile apps [8] and more.

In recent years, the advancement of electronics, information sciences and Artificial Intelligence supported the research and development in the telecommunication (TLC) field. In modern TLC systems, important aspects are flexibility and compatibility with multiple standards, often addressed with ML approaches. Some examples are Software Defined Radio (SDR) [9], Cognitive Radio (CR) [10] and Intelligent Radio systems (IR). IRs are capable to autonomously estimate the optimal communication parameters when the system operates in a time-variant environment. This intelligent behavior can be obtained by using conventional adaptive signal processing techniques or by using ML approaches.

In [11] the authors show a survey of different innovative ML techniques applied to telecommunications. In [12] the authors compare the modulation recognition performance using different ML techniques such as Logistic Regression (LR), Artificial Neural Networks (ANN) and Support Vector Machines (SVM) over different datasets (FSK and PSK-QAM signals). The authors in [13] propose the use of Deep Belief Networks in a demodulator architecture. Other noteworthy research papers in this context are [14]–[18]. Another important research field is related to the development of ML hardware accelerators that allow time and energy efficient ML algorithms execution [19]–[24].

ML techniques are usually classified in three main categories: Supervised, Unsupervised and Reinforcement Learning. The first two require a *training phase* to obtain an expert algorithm ready to be deployed in the field (*inference phase*). Supervised and unsupervised ML approaches rely on massive amounts of data, intensive offline training sessions and large parameter spaces [25]. Moreover, the inference performance

The associate editor coordinating the review of this manuscript and approving it for publication was Malik Jahan Khan.

degrades when the statistics of the input data differs from that of the training examples. In these cases, multiple training sessions are required to update the model [26]. In Reinforcement Learning (RL) the training and inference phases are not separated. The learner interacts with the environment to collect information and receives an immediate reward for each decision and action it takes. The reward is a numerical value that quantifies the quality of the action performed by the learner. Its aim is to maximize the reward while interacting with the environment through an iterative sequence of actions [27].

Consequently, in a scenario where the input data statistics evolves, a possible solution is the use of an RL approach. The main feature of RL is its capability to solve the task interacting with the environment by trail-and-error. For this reason, this methodology brings the following advantages:

- The off-line training phase is not required, avoiding the need for cumbersome training data-sets;
- Parameter space dimensions are generally smaller than in supervised and unsupervised approaches;
- The model is self-adapting using the data collected in the field.

RL became increasingly popular in robotics [28]–[31], Internet of Things (IoT) [32], financial trading [33] and industrial applications [34]. RL for multi-Agent systems is a growing research field as well. It is often bio-inspired [35] and the learners are organized in ensembles, i.e. swarms, to improve learning capabilities [36], [37].

However, the application of RL models to TLC is a fairly new and unexplored line of research. As detailed later, a conventional TLC receiver is a cascade of processing units based on tunable feedback loops and application-specific modules. For example, a Timing Recovery Loop (TRL) is a system that includes an error detector and a feedback loop filter to control the synchronization of the receiver resampler. Conventional approaches require the tuning of the filter parameters and error detectors that are specific to the transmission parameters, such as types of modulation scheme.

In this paper, we propose a TRL based on RL capable to operate independently of the modulation format, due to its adaptation capability. Moreover, a large training dataset is not required, differently from the cited researches about the use of supervised and unsupervised ML approaches in TLC. Our solution is based on Q-Learning, an RL algorithm developed by Watkins and Dayan [38]. This model learns from the input data how to synthesize the behavior of a symbol synchronizer by using a trial-and-error iterative process. The system performance has been validated by comparing the proposed approach to a standard synchronization method, the Mueller and Müller gate [39]. With respect to [40], we extend the method to a wider set of modulations schemes by using a modified and enhanced ML model.

The main advantages of our approach are:

- It is capable to operate on different modulation schemes (static conditions);
- It is capable of self adaptation to changes in the channel noise and modulation formats (changes in the environment);
- It avoids the loop filter parameter tuning phase;
- It avoids the need to allocate specific sub-channels for synchronization because the timing recovery is performed by using the input data.

In this paper, we prove that the use of a Reinforcement Learning based QAM/PSK symbol synchronizer is a valid timing recovery method by testing it in generic telecommunication system.

This paper is organized as follows. In Sect. II an overview about conventional symbol synchronization methods, RL and Q-Learning is provided. In Sect. III we illustrate the architecture and the sub-modules of the proposed RL based TRL. In Sect. IV we provide the experimental results regarding the choice of the Q-Learning hyperparameters and we compare the performance of the system to conventional synchronization methods. In Sect. V the conclusions are drawn.

II. BACKGROUND

In this section, we provide insights about conventional timing recovery methods and Reinforcement Learning.

A. SYMBOL SYNCHRONIZATION METHODS

A generic receiver front-end consists of a receive filter followed by a carrier and timing recovery system. The carrier recovery system is implemented with first or second-order Phase-Locked Loops (PLL). The symbol synchronization is performed using Timing Recovery Loops (TRL) as shown in [41], [42]. Examples of recovery loops are the Costas Loop, for carrier recovery [43], the Early-Late [42], [44], the Mueller and Müller [39], and the Gardner Loop [45] for symbol synchronization. The timing recovery is often implemented as a first-order TRL. A TRL consists of a timing phase detector (TPD), a loop filter, and a Numerically Controlled Oscillator (NCO) for resampling.

In this paper we compare the RL based synchronizer to the Mueller and Müller (M&M) method for two main reasons: the M&M recovery loop is a data-assisted timing synchronization algorithm and it uses only one sample per symbol. The M&M synchronizer (the BPSK version is shown in Fig. 1) resamples the signal $y[k]$ and computes the synchronization error $e[n] = x[n]a[n-1] - x[n-1]a[n]$. A decision device estimates the expected values $a[n]$ and $a[n-1]$, providing the constellation coordinates of the nearest symbol. The error signal drives a loop filter which controls the NCO that resamples the input waveform $y[k]$.

B. REINFORCEMENT LEARNING

Among the ML paradigms described in the Introduction, RL is the most appropriate method for symbol timing synchronization, because a lot of the new TLC applications imply dynamic scenarios (*environments*). RL is a ML methodology

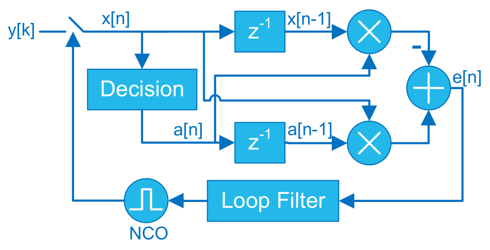


FIGURE 1. **Mueller and Müller timing recovery loop (BPSK implementation).**

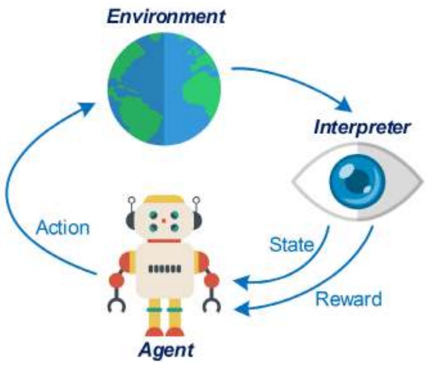


FIGURE 2. **Principle of reinforcement learning.**

used to train an object, called *Agent*, to perform a certain *task* by interacting with the *environment*. As shown in Fig. 2, the RL Agent adjusts its behavior by a trial-and-error process through a rewarding mechanism called *reinforcement* that is a measure of its task-solving overall capability [46]. The effects of the actions are gathered and estimated by an entity called *Interpreter* that computes the new state and rewards the Agent accordingly. The Interpreter can be external or internal to the Agent itself: in the latter case, the Agent and the Interpreter are merged into a more complex Agent able to sense the environment and capable of self-criticism. This is the type of Agent we use in this paper.

The proposed RL-based synchronizer interacts with the received signal generated by an evolving *TLC environment* and is capable to *decide* the best timing compensation value. This process takes place without the need for extensive training phases and large training datasets.

In standard RL algorithms, the environment is often modeled as a Markovian Decision Process (MDP) [46]. The Agent is characterized by the space $S \times A$, where S is the *state* vector and A is the set of the *actions*. The Agent evaluates the states $s_i \in S$ and *decides* to perform the action $a_i \in A$. The process by which the Agent senses the environment, collecting pieces of information and knowledge, is called *exploration*. At a certain time t, the Agent is in the state s_t and takes the action a_t. The *decision* to perform the action a_t takes place because the Agent iteratively learns and builds an optimal action-selection policy $\pi : S \times A$. Each $\pi(s, a)$ is a real number in the range $[0, 1]$. π is a probability map showing the chances of the Agent to take one of the actions in A given the current state s_t. Then, the Agent observes the effects generated by the action a_t on the environment.

The information derived from the environment response is called *reward*, a real number (positive or negative). Its value depends on how the action a_t brings the system closer to the task solution. The Agent aims to accumulate positive rewards and to maximize its expected return by iterating actions over time. This process is modeled by the *State-Value Function* $V_\pi(s)$ which is related to the current policy map and it is updated with the reward value at each iteration. Through $V_\pi(s)$ the Agent learns the *quality* of the new state s_{t+1} reached starting from s_t and executing a_t. The number of iterations taken by the Agent to complete the given task is called an *episode*. The agent reaches a terminal state s_{goal} and the updated State-Value Function is stored for further use. The subsequent episode starts with the updated State-Value Function available to the Agent. As $V_\pi(s)$ is updated, the Agent manages its set of actions in order to solve the task by maximizing the cumulative reward in fewer iterations. This process is called *exploitation*. The Agent explores and exploits the environment using the knowledge collected in the form of the π map.

In general, the environment is dynamic and the effect of an action in a certain state can change over time. By alternating exploration and exploitation, the RL Agent adapts its State-Value Function $V_\pi(s)$ over time following the environment changes. There are several RL algorithms that approximate and estimate the State-Value function, both for single and multi-Agent, and the most prominent example is Q-learning by Watkins and Dayan [38].

1) WATKINS' Q-LEARNING

Q-learning is an algorithm able to approximate the State-Value function $V_\pi(s)$ of an Agent in order to find an optimal action-selection policy π^*. As mentioned before, the Agent overall goal is to maximize its total accumulated future discounted reward derived from current and future actions. The State-Value function is defined as $V_\pi(s) = E[R]$ and R is the *Discounted Cumulative Future Reward* defined as (1):

$$R = \sum_{t=0}^{\infty} \gamma^t r_t \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor and r_t is the reward at time t. In [38] the authors demonstrate that to approximate the State-Value $V_{\pi^*}(s)$ function related to the optimal policy π^* it is sufficient to compute a map called *Action-Value Matrix* $Q_{\pi^*} : S \times A$. The elements of Q represent the quality of the related policy $\pi(s_t, a_t)$. The Q values are updated at every iteration (time t) using the Q-Learning update algorithm in the form of the Bellman equation [47] and reported below (2).

$$\begin{aligned} Q(s_t, a_t) \leftarrow\ & (1-\alpha)Q(s_t, a_t) \\ & + \alpha \left(r_t + \gamma \max_a Q(s_{t+1}, a)\right) \end{aligned} \tag{2}$$

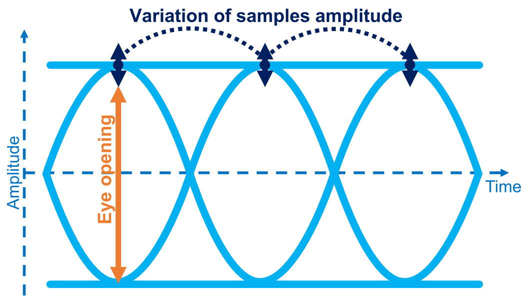


FIGURE 3. Objectives: minimization of the amplitude variation and maximization of the eye-opening (BPSK waveform).

where α is the learning rate, γ is the discount factor and r_t is the reward at time t. Some state-of-the-art action-selection policies widely used in Reinforcement Learning frameworks [46] are:

- *Random*: the Agent selects an action $a_i \in A$ with a random approach, ignoring the Q values;
- *Greedy*: the Agent selects the action with the highest corresponding Q value;
- ϵ*-Greedy*: the chance of taking a random action is a value $\epsilon \in [0, 1]$ and the probability of a greedy choice is $1 - \epsilon$;
- *Boltzmann*: the Q values are used to weight the action-selection probabilities according to the Boltzmann distribution based on the temperature parameter τ.

In the case of an Agent required to perform multiple actions simultaneously, it is possible to apply the concept of *Action Branching* [48] that extends the dimensionality of the action space.

III. TIMING SYNCHRONIZER BASED ON Q-LEARNING

In this section, we present the design of a TRL based on Q-Learning able to recover the timing of a raised-cosine-shaped signal when QAM and PSK modulation schemes are used. In our system, the input signal $y[k]$ is downsampled by a factor M obtaining the signal $x[n]$ (as for the Mueller and Müller gate). The Agent task is to compensate for the symbol timing error by delaying or anticipating the resampling time. In our experiments, the downsampling factor M is equal to the number of samples per symbol. The definitions of the Agent state is related to a measure on the resampled symbol (detailed in Sect. III-A.2). The Agent action is defined as a double control (Sect III-C): the first one is used to manage the resampling timing and the second one to modify the input signal amplitude.

With respect to Fig. 3, the Agent is required to fulfill two main objectives:

1) To minimize the variation in amplitude of consecutive resampled symbols $x[n]$ with respect to an expected constellation (dark blue arrows);
2) To maximize the eye-opening (orange arrow).

With respect to [40], the algorithm is extended to support the QAM synchronization.

The proposed RL Agent architecture is illustrated in Fig. 4 and it consists of four main sub-modules:

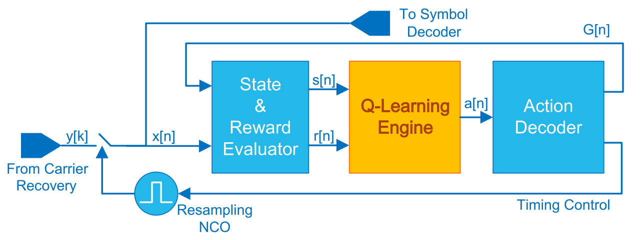


FIGURE 4. Q-Learning timing synchronizer architecture.

1) *State and Reward Evaluator Block*: it is the Interpreter in the RL framework;
2) *Q-Learning Engine*: it updates the Q matrix and implements the action-selection policy π;
3) *Action Decoder*: it carries out the Agent decisions to manage the signal sampling delay and its amplitude;
4) *Numerically Controlled Oscillator*: it provides the resampling timing.

Additional details and accurate descriptions of the TRL blocks are provided in the following subsections.

A. STATE AND REWARD EVALUATOR

The basic idea behind the RL based TRL is illustrated in Fig.5. In this figure the input $y[k]$ is a BPSK signal. The variation range of the resampled signal absolute value $|(x[n])|$ is represented by the pink band (Fig. 5a) and green (Fig. 5b). The Agent goal is the minimization of this variation and, in addition to this, it minimizes the difference between the amplitude of the resampled signal $x[n]$ (red crosses) and the expected symbol value x^*(blue line, the value is 1 in the depicted example). We design an RL Interpreter, called State and Reward Evaluator (SRE), that processes $x[n]$ to fulfill these objectives.

We design the Agent states $s_i \in S$ and reward r depending on the input signal amplitude and its variation over time. As depicted in Fig. 6, the SRE module is built as two blocks in cascade called *Constellation Conditioning* and *State & Reward Computing*.

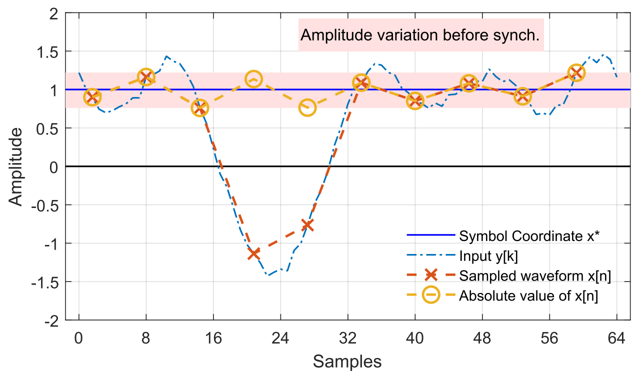
1) CONSTELLATION CONDITIONING

The main limitation of the approach proposed in [40] is the inability to synchronize QAM signals. To address this problem we use a coordinate transformation approach, implemented as shown in Fig.7.

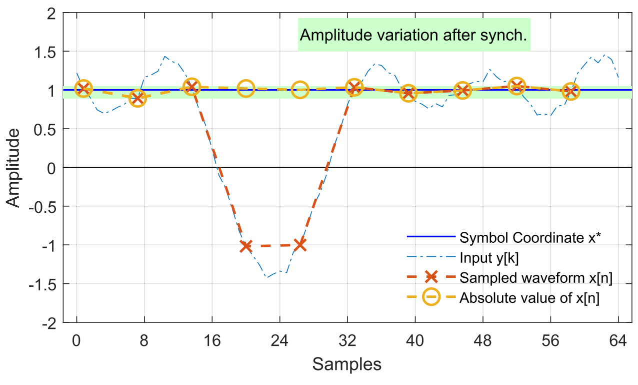
The following equation represents the constellation conditioning algorithm:

$$\hat{x} = |[G \cdot x_r - \text{round}(G \cdot x_r)] + 1| \qquad (3)$$

where $\hat{x}[n]$ is the conditioned symbol, x_r is the real part of $x[n]$, and the gain value $G[n]$ is controlled by the Agent with the action $a^2[n]$, detailed later. Using this approach, the transformed signal $\hat{x}[n]$ tends to an optimal value $\hat{x}^*$ that depends on the value of $G[n]$. The effect of the *wrap* block is modeled by $[G \cdot x_r - \text{round}(G \cdot x_r)]$. The constellation conditioning process is depicted in Fig. 8. In this example, we consider



(a)



(b)

FIGURE 5. Resampling of a BPSK waveform ($M = 8$): (a) before synchronization, (b) after synchronization.

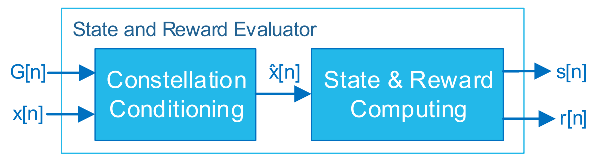


FIGURE 6. State and reward evaluator (SRE) block.

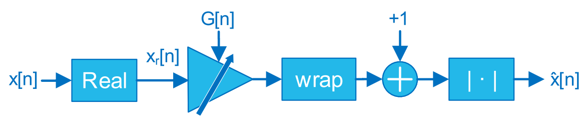


FIGURE 7. Constellation conditioning block.

a 16QAM constellation where the coordinates of $x[n]$ are projected on the real axis, aligned using the gain block, collapsed and merged on the expected symbol coordinate $\hat{x}^*$.

2) STATE & REWARD COMPUTING

In Fig. 9 the State and Reward Computing block is shown.

The state function (4) and the reward function (5) are:

$$s[n] = N/2 \cdot (\nabla^2\hat{x}[n] + 1) + \hat{x}[n-2] \tag{4}$$

$$r[n] = \left[\hat{x}[n-2] - |\nabla^2\hat{x}[n]|\right]^q \tag{5}$$

where N is the number of states and q must be an odd integer to keep the sign. $\nabla^2\hat{x}[n]$ is the second-order finite difference of $\hat{x}[n]$, as shown in (6):

$$\nabla^2\hat{x}[n] = \hat{x}[n] - 2\hat{x}[n-1] + \hat{x}[n-2] \tag{6}$$

To compute the state $s[n]$, the value of $\nabla^2\hat{x}[n]$ is scaled by $N/2$ to be compatible with the coordinates of the Agent Q-matrix. If the Agent retrieves the correct symbol phase, the differential $\nabla^2\hat{x}[n]$ tends to 0 and $\hat{x}[n]$ to $\hat{x}^*$. For this reason, according to (4), the target state s_i^* is placed in the neighborhood of $s_i = N/2 + \hat{x}^*$ and the reward, in (5), is maximized.

B. Q-LEARNING ENGINE

The purpose of this module is to update the Action-Value elements of Q using equation (2). As shown in Fig. 4, the inputs are the Agent state $s[n]$ and the reward $r[n]$. The output is the action $a[n]$, determined using one of the policies described in Sect. II-B.1. In particular, the action is branched in two components:

- *Branch* $a^1 = \{a_0^1, a_1^1, a_2^1\}$ represents the action vector to *delay*, *stop* or *anticipate* the resampler;
- *Branch* $a^2 = \{a_0^2, a_1^2, a_2^2\}$ is an action vector used to *increase*, *hold* or *decrease* $G[n]$.

The Agent action space $A = a^1 \times a^2$ is bi-dimensional and it is processed and actuated by the Action Decoder block, detailed in the next Section. The mapping of the state-action space $S \times A$ into Q is a tensor with one dimension for the states and two for the actions, as illustrated in Fig. 10.

There are $3 \times 3 = 9$ available actions for each state (s_i). The hyperparameter $N \in \mathbb{N}$ is the cardinality of the state space S (hence $s_i \in \mathbb{N}$). For this reason, the state values $s[n]$ calculated in (4) are rounded to address the Q-tensor.

C. ACTION DECODER

The Action Decoder implements the decisions taken by the Agent. The action $a = \{a^1, a^2\}$ is the input of this block. The outputs are used by the Agent to control the NCO timing and $G[n]$ respectively. This block computes the increments to be assigned to the NCO timing and the Gain $G[n]$.

The action vector a^1 control the resampler using three different actions:

- $a_0^1 = 0$: the Agent *decides* that the current timing is correct;
- $a_1^1 = +1$: the Agent anticipates the resampling;
- $a_2^1 = -1$: the Agent delays the resampling.

The purpose of the action a^2 is to modify the amplitude of the input signal, as shown in Fig. 8 and discussed in Sect III-A. The elements of a^2 are the increments used by the Action Decoder to update the input gain $G[n]$:

- $a_0^2 = 0$: the current $\hat{x}^*$ value is appropriate and the Agent decides to maintain the $G[n]$ constant;
- $a_1^2 = +0.02$: $G[n]$ is increased;
- $a_2^2 = -0.02$: $G[n]$ is decreased.

The initial control values for the NCO timing and $G[n]$ are 0 and 1 respectively.

IV. EXPERIMENTS AND RESULTS

In this section, we present three experiments. In the first one we find suitable Q-Learning hyperparameters, in the second

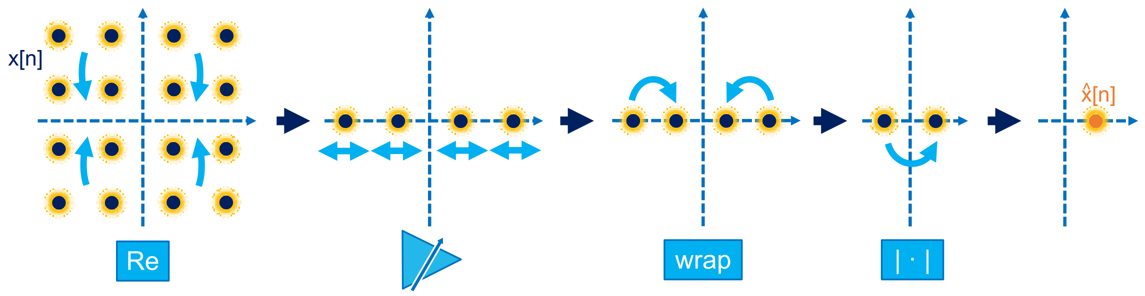


FIGURE 8. Example of constellation conditioning on a 16QAM modulation scheme.

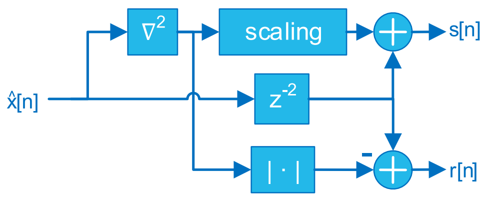


FIGURE 9. State and reward computing block.

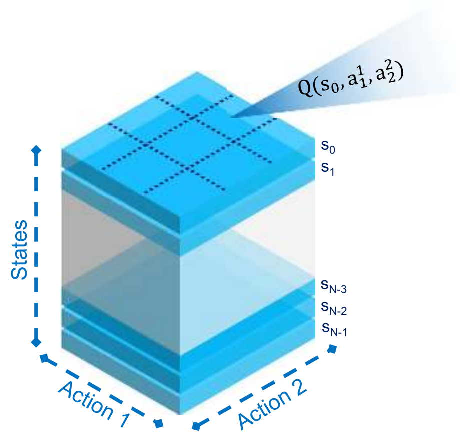


FIGURE 10. Mapping of the state-action space into *Q* values.

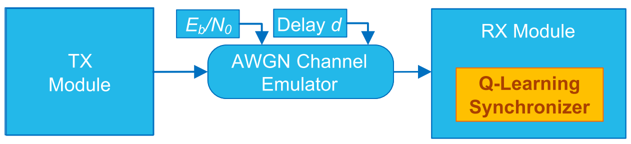


FIGURE 11. Experimental setup used to test and characterize the proposed Q-Learning TRL.

one we test the adaptivity properties of the Q-Learning Agent, and, in the last one, we compare our algorithm to the Mueller and Müller Loop when signals with different modulation formats in a generic telecommunication system are used. The experimental setup is common among the experiments and it is shown in Fig. 11.

The TX-Module is a base-band transmitter in which the modulation scheme and Root-Raised Cosine (RRC) shaping filter are parametric. The AWGN Channel Emulator is configurable in delay and noise power (measured in terms of E_b/N_0). The RX-Module includes an RRC receive filter and the Q-Learning Synchronizer. In these experiments, we assume a recovered carrier. The performance of the synchronizer is evaluated in terms of Modulation Error Ratio (MER), defined as in [49].

In the first experiment, we asses the optimal Q-Learning hyperparameters a posteriori. At the same time, we aim to reduce the Agent complexity, i.e. the dimensionality of the hyperparameters space. In the second experiment, we change the environment properties i.e. the modulation format to evaluate the Q-Learning Agent adaptation capability. In the last experiment, we compare the MER of the Q-Learning synchronizer with that obtained by the Mueller and Müller method for PSK and QAM modulation schemes. The experiments are characterized by the following Q-Learning hyperparameters (the values that were validated a posteriori) and TLC parameters:

- The $N \times 3 \times 3$ Q-Tensor is initialized with all zero values;
- The Agent number of states is $N = 64$;
- The values of α and γ in (2) are 0.1 and 0.01, respectively;
- The Agent action-selection policy is ϵ-*greedy* with $\epsilon = 10^{-5}$;
- The samples per symbol of the transmitted waveform is equal to $M = 32$.

The MER measurements, if not explicitly otherwise indicated, are obtained as the average of 200 simulations where we select the last 200 symbols in steady-state condition. The transmitted PSK symbols have a constant amplitude equal to 1. In QAM mode, the symbols have a fixed minimum distance of 2.

A. HYPERPARAMETERS ANALYSIS

Considering equation (2) and the Agent features, the hyperparameters are:

- *Action-selection policy* π: it represents the action choice rule according to the Q-values. In our work we employed ϵ-Greedy with $\epsilon = 10^{-5}$ to facilitate the state-space exploration. Moreover, it is important to avoid the local maxima of the reward function.
- *Learning rate* α: it defines the convergence speed of the Q matrix towards the ideal State-Value function. In the case of a dynamic environment, it also defines the adaptation speed of the Agent. For $\alpha = 0.1$, we found

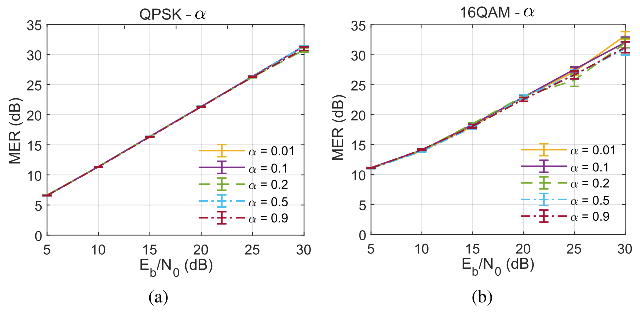
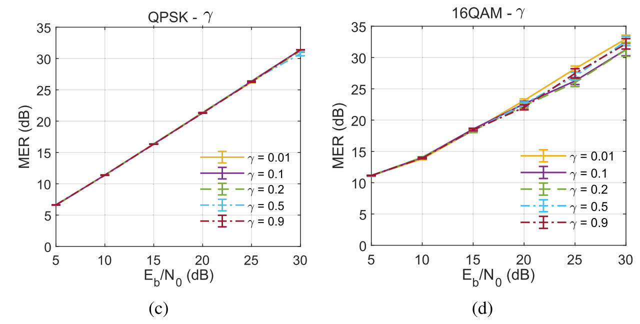



FIGURE 12. MER variation as a function of α and γ for different E_b/N_0 and different modulation formats.

a good trade-off between algorithm convergence speed and local maxima avoidance.

- *Discount factor* γ: it weights the outcome of the present actions with respect to the future ones. A suitable γ value is 0.01.
- *Number of states* N: it defines the state-space quantization, hence the Q-tensor dimension (Fig. 10).

In the following analysis, different simulations have been implemented for PSK and QAM formats, and in the next Sections the QPSK and 16QAM are illustrated.

1) MER VARIATION AS A FUNCTION OF α AND γ

To study the range of variation of the Q-Learning TRL hyper-parameters with respect to the modulation format and E_b/N_0 we set an initial constant timing delay $d = M/4$, where M is the number of samples per symbol. The graphs in Fig. 12 show that the variations of α (Fig. 12a and Fig. 12b) and γ (Fig. 12c and Fig. 12d) do not affect the MER performance significantly. Consequently, the default set $\alpha = 0.1$ and $\gamma = 0.01$ is a proper design choice for different modulation formats.

2) NUMBER OF STATES N

The number of states determines the environment observation precision of the Agent. To analyze the system MER performance, we simulate our system with N in the range 8-1024 and $E_b/N_0 = 5 - 30$ dB for PSK and QAM modulation schemes. The MER results are evaluated after 1000 exploration symbols.

Figure 13a shows that the MER is affected by N only in the case of high SNR. In this case, the MER grows when N increases (10 dB of spread at $E_b/N_0 = 30$ dB). In the

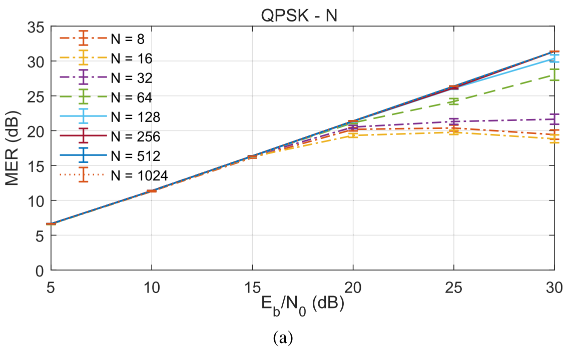
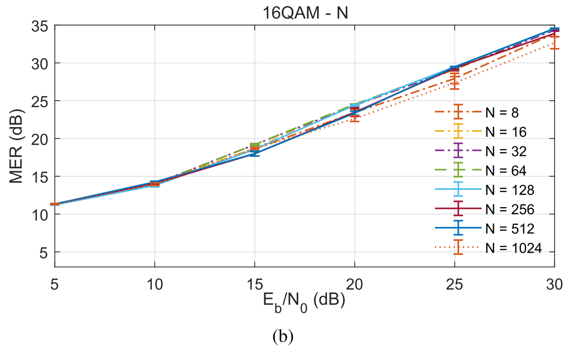



FIGURE 13. Variation of N. MER measured for different channel noise levels.

16QAM case in Fig. 13b, we observe the same dependence with respect to N but with a smaller spread (2.5 dB at $E_b/N_0 = 25 - 30$ dB).

This experiment allows a proper selection of the number of states N. In the QPSK case, the MER curves are monotonically growing for $N \geq 64$, consequently there is no advantage to use an Agent with more than 64 states. In the 16QAM tests, the curves show a similar trend. The best trade-off in terms of MER and number of states is $N = 64$. The resulting Q-Tensor consists of $64 \times 3 \times 3 = 576$ elements.

B. Q-LEARNING TRL ADAPTIVITY EXPERIMENT

As mentioned in Sect.I and Sect. II-B, an RL Agent is able to adapt itself to the evolution of the environment. This property ensures high system availability on the field and no needs for training sessions. With the following experiment we show the adaptivity properties that the RL approach provides and how the proposed TRL is able to operate independently of the modulation scheme.

In Fig. 14, the initial timing delay is set to $d = 7$ (in samples, horizontal dashed gray line) and the modulation format changes from QPSK to 16QAM (vertical red line). This change in the modulation scheme models an environment evolution and it is useful to observe the Agent behavior in terms of adaptivity.

The Agent manages its actions to carry out the best TRL controls, i.e. the timing compensation (blue curve) and the value of G (orange curve). The Agent experiences two exploration phases (green background) and two exploitation phases (white background). After the first exploration phase

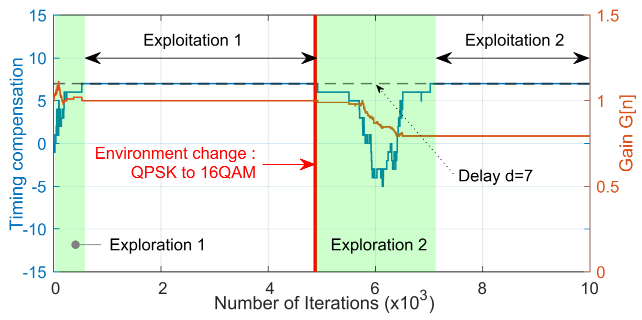


FIGURE 14. Q-Learning adaptivity: Timing compensation and gain *G*[*n*].

(∼ 500 iterations), the Agent *decides* that the best timing compensation and gain values are 7 and ∼1, in a QPSK environment. The values of the Q-matrix converge to the optimal ones and the Agent begins to exploit its knowledge by maintaining the optimal TRL controls.

At a certain point, the environment changes from QPSK to 16QAM. Because of that, the Agent is forced to adapt itself by entering the second exploration phase to update its Q-matrix. After about 2000 Q-learning iterations, the Agent enters the last exploitation phase and it *decides* to set the timing compensation correctly to 7 and the gain to ∼ 0.7. The modification of $G[n]$ from ∼1 to ∼0.7 is related to the change of the real part of the coordinate of the received symbol from the QPSK to the QAM format.

C. MODULATION SCHEME EXPERIMENTS AND COMPARISON WITH CONVENTIONAL METHODS

In this experiment we test the Q-Leaning synchronizer performance when different modulation schemes are used. Moreover, we compare its performance to conventional synchronization methods. The Q-Learning hyperparameters are $N = 64$, $\alpha = 0.1$, $\gamma = 0.01$, values suitable for BPSK, QPSK, 16QAM, 64QAM and 256QAM. The channel delay is constant and equal to $d = M/4$, where M is the number of samples per symbol. The Q-Learning synchronizer is compared to the following methods:

- A *Reference ideal resampler*: a resampler with the exact timing compensation for the channel delay d;
- A *Mueller and Müller timing recovery loop*: the state-of-the-art TRL discussed in Sec. II-A.

The graphs in Fig. 15 are obtained with the same set of the Q-Learning hyperparameters. The Mueller and Müller TRL filter parameters depend from the modulation method (one set for PSK modulations and one for QAM) and they are reported in Table 1.

Figure 15 and Table 2 show the experimental results. The number of simulations is 200, the number of exploration

TABLE 1. Loop filter parameters of mueller and Müller TRLs.

Parameter	PSK	QAM
Damping Factor	1	2
Normalized Loop Bandwidth	0.1	0.005
Phase Detector Gain	2.7	2.7

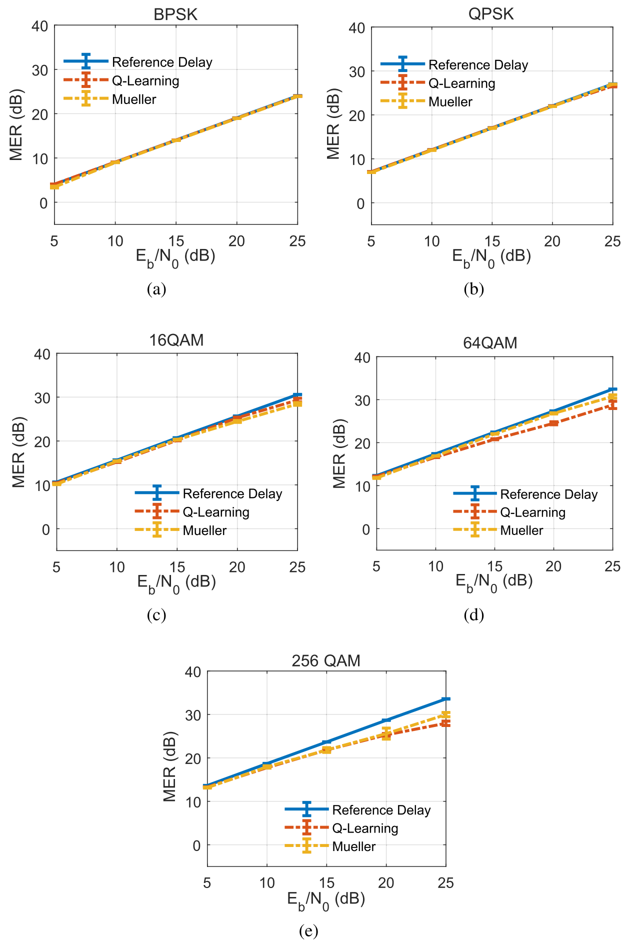


FIGURE 15. MER for different modulation schemes and comparison with the Reference ideal resampler and the Mueller and Müller approach.

symbols is 10000 and the MER of a single simulation is computed over 200 symbols. Each value in Table 2 is the average MER plus or minus its standard deviation.

Figures 15a and 15b show the MER measurements for BPSK and QPSK respectively. The MER values for BPSK and QPSK are similar. The only exception is in the QPSK configuration as the Q-Learning synchronizer loses up to 0.7dB at $E_b/N_0 = 25$ dB with respect to the reference model.

The QAM related graphs (Fig. 15c, Fig. 15d and Fig. 15e) show an interesting performance. In the case of high noise ($E_b/N_0 = 5 - 10$ dB), the three systems perform similarly. In the mid-range ($10 - 20$ dB), the Q-Learning performance is similar to the Mueller and Müller loop in the 16QAM and 256QAM. In the same range, for 64QAM, we measured MER values lower (1.5 dB on average) than the Mueller and Müller loop. In the $20 - 25$ dB range, the Q-Learning synchronizer is outperformed by about 2 dB due to a poorer efficacy of the exploration phase of the Agent. This leads to the conclusion that the presence of white noise in the RL environment forces the Agent to explore more states, thus converging to its optimal Q-matrix much more efficiently. Taking into account all MER measurements, the average difference between our method and the conventional TRL is lower than 1 dB.

TABLE 2. Comparison of the Q-Learning synchronizer (*QLearn*) with the reference ideal sampler (*Ref*) and the mueller and Müller timing recovery loop (*M&M*).

E_b/N_0	System	BPSK MER (dB)	QPSK MER (dB)	16QAM MER (dB)	64QAM MER (dB)	256QAM MER (dB)
5dB	Ref	4.03 ± 0.00	7.05 ± 0.00	10.64 ± 0.03	12.35 ± 0.04	13.65 ± 0.02
	M&M	3.43 ± 0.19	6.91 ± 0.02	10.02 ± 0.12	11.89 ± 0.11	13.16 ± 0.10
	QLearn	4.00 ± 0.01	7.03 ± 0.02	10.44 ± 0.04	12.19 ± 0.06	13.30 ± 0.05
10dB	Ref	9.04 ± 0.00	12.05 ± 0.00	15.63 ± 0.07	17.42 ± 0.04	18.69 ± 0.05
	M&M	8.97 ± 0.02	11.94 ± 0.02	15.25 ± 0.08	17.00 ± 0.05	17.92 ± 0.26
	QLearn	9.01 ± 0.01	12.03 ± 0.01	15.22 ± 0.11	16.62 ± 0.09	17.75 ± 0.09
15dB	Ref	14.04 ± 0.00	17.05 ± 0.00	20.72 ± 0.03	22.46 ± 0.03	23.66 ± 0.04
	M&M	13.96 ± 0.02	16.95 ± 0.02	20.17 ± 0.09	22.14 ± 0.07	21.81 ± 0.55
	QLearn	14.02 ± 0.01	17.02 ± 0.01	20.17 ± 0.12	20.80 ± 0.12	21.82 ± 0.16
20dB	Ref	19.04 ± 0.00	22.05 ± 0.00	25.65 ± 0.06	27.37 ± 0.04	28.67 ± 0.04
	M&M	18.95 ± 0.02	21.96 ± 0.02	24.30 ± 0.21	26.91 ± 0.14	25.59 ± 1.27
	QLearn	19.01 ± 0.03	22.02 ± 0.03	25.30 ± 0.15	24.51 ± 0.36	25.30 ± 0.31
25dB	Ref	24.04 ± 0.00	27.05 ± 0.00	30.59 ± 0.03	32.45 ± 0.03	33.58 ± 0.04
	M&M	23.91 ± 0.02	26.97 ± 0.01	28.29 ± 0.30	30.84 ± 0.39	29.96 ± 0.49
	QLearn	23.94 ± 0.07	26.56 ± 0.20	29.33 ± 0.42	28.78 ± 0.84	27.96 ± 0.51

V. CONCLUSION

In this paper, we presented an innovative method based on RL to implement a QAM/PSK symbol synchronizer. This approach employs an RL Agent able to synthesize a TRL behavior compatible with a number of modulation schemes.

The RL algorithm used to approximate the State-Value function is Q-Learning. The state and reward are computed from the sampled symbol amplitude $\hat{x}[n]$ and its differential $\nabla^2\hat{x}[n]$. The Agent action is branched in two sub-action: the first one to manage the timing and the second one to adjust the input signal amplitude. The proposed Q-Learning TRL includes four modules: a State and Reward Evaluator (the RL Interpreter), a Q-Learning Engine, an Action decoder, and an NCO. The compatibility with respect to the different modulation schemes is obtained through the *Constellation Conditioning* block. This solution extends the work presented in [40] to QAM formats.

The proposed method was validated through the simulation of a base-band transmitter, receiver, and an AWGN channel emulator, with E_b/N_0 in the range 5-30 dB through three experiments. In the first one, we estimated a set of hyperparameters for BPSK, QPSK, 16QAM, 64QAM and 256QAM. The performance of the RL synchronizer is evaluated in terms of Modulation Error Ratio (MER).

In the second experiment, we proved the autonomous adaptation capability of our approach, suitable for the new intelligent communication systems. In the last experiment, we compared the RL approach to the Mueller and Müller TRL: for BPSK and QPSK modulations, the two synchronizers achieve similar MER values, close to a Reference resampler. The proposed TRL is slightly outperformed for the 64QAM and 256QAM in low-noise scenarios ($E_b/N_0 \geq$ 15 dB). The RL synchronizer performance is the same as the Mueller and Müller TRL for 16QAM in the full noise range.

These results are a good trade-off between flexibility and performance. Moreover, unlike conventional methods, the Q-Learning based TRL is able to adapt itself to evolving scenarios, avoiding parameter tuning. We plan to expand our research to support additional modulation formats and to implement suitable hardware architectures.

REFERENCES

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[2] M. Chen, Y. Hao, K. Hwang, L. Wang, and L. Wang, "Disease prediction by machine learning over big data from healthcare communities," *IEEE Access*, vol. 5, pp. 8869–8879, 2017.

[3] D. Cabrera, C. Cubillos, A. Cubillos, E. Urra, and R. Mellado, "Affective algorithm for controlling emotional fluctuation of artificial investors in stock markets," *IEEE Access*, vol. 6, pp. 7610–7624, 2018.

[4] A. L'Heureux, K. Grolinger, H. F. Elyamany, and M. A. M. Capretz, "Machine learning with big data: Challenges and approaches," *IEEE Access*, vol. 5, pp. 7776–7797, 2017.

[5] G. Wang, "A perspective on deep imaging," *IEEE Access*, vol. 4, pp. 8914–8924, 2016.

[6] K. Muhammad, J. Ahmad, I. Mehmood, S. Rho, and S. W. Baik, "Convolutional neural networks based fire detection in surveillance videos," *IEEE Access*, vol. 6, pp. 18174–18183, 2018.

[7] T. Petrovic, K. Echigo, and H. Morikawa, "Detecting presence from a WiFi router's electric power consumption by machine learning," *IEEE Access*, vol. 6, pp. 9679–9689, 2018.

[8] A. Sehgal and N. Kehtarnavaz, "A convolutional neural network smartphone app for real-time voice activity detection," *IEEE Access*, vol. 6, pp. 9017–9026, 2018.

[9] M. Dillinger, K. Madani, and N. Alonistioti, *Software Defined Radio: Architectures, Systems and Functions*. Hoboken, NJ, USA: Wiley, 2005.

[10] J. Mitola, "Cognitive Radio—An integrated agent architecture for software defined radio," Ph.D. dissertation, Dept. Teleinform., KTH Roy. Inst. Technol., Stockholm, Sweden, 2000.

[11] A. He, K. K. Bae, T. R. Newman, J. Gaeddert, K. Kim, R. Menon, L. Morales-Tirado, J. J. Neel, Y. Zhao, J. H. Reed, and W. H. Tranter, "A survey of artificial intelligence for cognitive radios," *IEEE Trans. Veh. Technol.*, vol. 59, no. 4, pp. 1578–1592, May 2010.

[12] M. Bari, H. Taher, S. S. Sherazi, and M. Doroslovacki, "Supervised machine learning for signals having RRC shaped pulses," in *Proc. 50th Asilomar Conf. Signals, Syst. Comput.*, Nov. 2016, pp. 652–656.

[13] M. Fan and L. Wu, "Demodulator based on deep belief networks in communication system," in *Proc. Int. Conf. Commun., Control, Comput. Electron. Eng. (ICCCCEE)*, Jan. 2017, pp. 1–5.

[14] Q. Chen, Y. Wang, and C. W. Bostian, "Universal classifier synchronizer demodulator," in *Proc. IEEE Int. Perform., Comput. Commun. Conf.*, Dec. 2008, pp. 366–371.

[15] M. Zhang, Z. Liu, L. Li, and H. Wang, "Enhanced efficiency BPSK demodulator based on one-dimensional convolutional neural network," *IEEE Access*, vol. 6, pp. 26939–26948, 2018.

[16] T. A. Chadov, S. D. Erokhin, and A. I. Tikhonyuk, "Machine learning approach on synchronization for FEC enabled channels," in *Proc. Syst. Signal Synchronization, Generating Process. Telecommun. (SYNCHROINFO)*, Jul. 2018, pp. 1–4.

[17] J. J. G. Torres, A. Chiuchiarelli, V. A. Thomas, S. E. Ralph, A. M. C. Soto, and N. G. González, "Adaptive nonsymmetrical demodulation based on machine learning to mitigate time-varying impairments," in *Proc. IEEE Avionics Vehicle Fiber-Opt. Photon. Conf. (AVFOP)*, Oct./Nov. 2016, pp. 289–290.

[18] Y. Liu, Y. Shen, L. Li, and H. Wang, "FPGA implementation of a BPSK 1D-CNN demodulator," *Appl. Sci.*, vol. 8, no. 3, p. 441, 2018.

[19] G. C. Cardarilli, L. D. Nunzio, R. Fazzolari, D. Giardino, M. Matta, M. Re, F. Silvestri, and S. Spanò, "Efficient ensemble machine learning implementation on FPGA using partial reconfiguration," in *Proc. Int. Conf. Appl. Electron. Pervading Ind., Environ. Soc.*, in Lecture Notes in Electrical Engineering, vol. 550. Berlin, Germany: Springer, 2018, pp. 253–259.

[20] D. Giardino, M. Matta, M. Re, F. Silvestri, and S. Spanò, "Ip generator tool for efficient hardware acceleration of self-organizing maps," in *Proc. Int. Conf. Appl. Electron. Pervading Ind., Environ. Soc.*, in Lecture Notes in Electrical Engineering, vol. 550. Berlin, Germany: Springer, 2018, pp. 493–499.

[21] L. M. D. Da Silva, M. F. Torquato, and M. A. C. Fernandes, "Parallel implementation of reinforcement learning Q-learning technique for FPGA," *IEEE Access*, vol. 7, pp. 2782–2798, 2018.

[22] R. Li, Z. Zhao, Q. Sun, C.-L. I, C. Yang, X. Chen, M. Zhao, and H. Zhang, "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74429–74441, 2018.

[23] C. Savaglio, P. Pace, G. Aloi, A. Liotta, and G. Fortino, "Lightweight reinforcement learning for energy efficient communications in wireless sensor networks," *IEEE Access*, vol. 7, pp. 29355–29364, 2019.

[24] R. Ali, N. Shahin, Y. B. Zikria, B.-S. Kim, and S. W. Kim, "Deep reinforcement learning paradigm for performance optimization of channel observation–based MAC protocols in dense WLANs," *IEEE Access*, vol. 7, pp. 3500–3511, 2018.

[25] H. Kim, H. Nam, W. Jung, and J. Lee, "Performance analysis of CNN frameworks for GPUs," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2017, pp. 55–64.

[26] A. Ray, *Compassionate Artificial Intelligence: Frameworks and Algorithms*. CA, USA: Compassionate AI Lab, 2018.

[27] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. Cambridge, MA, USA: MIT Press, 2018, pp. 7–8.

[28] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1334–1373, 2015.

[29] A. Konar, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved Q-learning for path planning of a mobile robot," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 43, no. 5, pp. 1141–1153, Sep. 2013.

[30] J.-L. Lin, K.-S. Hwang, W.-C. Jiang, and Y.-J. Chen, "Gait balance and acceleration of a biped robot based on Q-learning," *IEEE Access*, vol. 4, pp. 2439–2449, 2016.

[31] S. Wu, "Illegal radio station localization with UAV-based Q-learning," *China Commun.*, vol. 15, no. 12, pp. 122–131, Dec. 2018.

[32] J. Zhu, Y. Song, D. Jiang, and H. Song, "A new deep-Q-learning-based transmission scheduling mechanism for the cognitive Internet of Things," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2375–2385, Aug. 2018.

[33] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 653–664, Mar. 2017.

[34] C. Wei, Z. Zhang, W. Qiao, and L. Qu, "Reinforcement-learning-based intelligent maximum power point tracking control for wind energy conversion systems," *IEEE Trans. Ind. Electron.*, vol. 62, no. 10, pp. 6360–6370, Oct. 2015.

[35] I. Navarro and F. Matía, "An introduction to swarm robotics," *ISRN Robot.*, vol. 2013, Jun. 2012, Art. no. 608164.

[36] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: A review from the swarm engineering perspective," *Swarm Intell.*, vol. 7, no. 1, pp. 1–41, Mar. 2013.

[37] M. Matta, G. C. Cardarilli, L. D. Nunzio, R. Fazzolari, D. Giardino, F. Silvestri, and S. Spanò, "Q-RTS: A real-time swarm intelligence based on multi-agent Q-learning," *Electron. Lett.*, vol. 15, no. 10, pp. 589–591, May 2019.

[38] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[39] K. Mueller and M. Muller, "Timing recovery in digital synchronous data receivers," *IEEE Trans. Commun.*, vol. 24, no. 5, pp. 516–531, May 1976.

[40] G. C. Cardarilli, L. D. Nunzio, R. Fazzolari, D. Giardino, M. Matta, F. Silvestri, M. Re, and S. Spanò, "A Q-learning based PSK symbol synchronizer," in *Proc. Int. Symp. Signals, Circuits Syst. (ISSCS)*, Jul. 2019, pp. 1–4.

[41] J. R. Barry, E. A. Lee, and D. G. Messerschmitt, *Digital Communication*, 3rd ed. Springer, 2012, pp. 137–139 and 727–760.

[42] F. Ling and J. Proakis, *Synchronization in Digital Communication Systems*. Cambridge, U.K.: Cambridge Univ. Press, 2017.

[43] J. P. Costas, "Synchronous communications," *Proc. IRE*, vol. 44, no. 12, pp. 1713–1718, Dec. 1956.

[44] B. Sklar, *Digital Communications: Fundamentals and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, 1988.

[45] F. Gardner, "A BPSK/QPSK timing-error detector for sampled receivers," *IEEE Trans. Commun.*, vol. 34, no. 5, pp. 423–429, May 1986.

[46] R. S. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[47] R. Bellman, *Dynamic Programming*, 1st ed. Princeton, NJ, USA: Princeton Univ. Press, 1957.

[48] A. Tavakoli, F. Pardo, and P. Kormushev, "Action branching architectures for deep reinforcement learning," in *Proc. 32nd AAAI Conf. Artif. Intell.*, Apr. 2018, pp. 4131–4138.

[49] *Digital Video Broadcasting (DVB); Measurement Guidelines for DVB Systems*, document ETR 290, ETSI, 1997, vol. 14, pp. 43–44.

MARCO MATTA was born in Cagliari, Italy, in 1989. He received the B.S. and the M.S. degrees in electronic engineering from the University of Rome Tor Vergata, Italy, in 2014 and 2017, respectively, where he is currently pursuing the Ph.D. degree in electronic engineering. Since 2017, he has been a member of the DSPVLSI Research Group of the University of Rome Tor Vergata. His current research interest includes the development of hardware platforms and low-power accelerators aimed machine learning algorithms and telecommunications. In particular, he is currently focused on the implementation of reinforcement learning models on FPGA.

GIAN CARLO CARDARILLI (S'79–M'81) was born in Rome, Italy. He received the laurea degree *(summa cum laude)* from the University of Rome La Sapienza, in 1981. He has been with the University of Rome Tor Vergata, since 1984. He is currently a Full Professor of digital electronics and electronics for communication systems with the University of Rome Tor Vergata. From 1992 to 1994, he was with the University of L'Aquila. From 1987 to 1988, he was with the Circuits and Systems Team, EPFL, Lausanne, Switzerland. His current research interest includes VLSI architectures for signal processing and IC design. He published over than 160 articles in international journals and conferences in this field. He has also regular cooperation with companies as: Alcatel Alenia Space, Italy; STM, Agrate Brianza, Italy; Micron, Italy; and Selex S.I., Italy. His scientific interest concerns the design of special architectures for signal processing. He is involved in the field of computer arithmetic and its application to the design of fast signal digital processor.

LUCA DI NUNZIO received the master's degree *(summa cum laude)* in electronics engineering and the Ph.D. degree in systems and technologies for the space from the University of Rome Tor Vergata, in 2006 and 2010, respectively. He has a working history with several companies in the fields of electronics and communications. He is currently an Adjunct Professor with the Digital Electronics Laboratory, University of Rome Tor Vergata and an Adjunct Professor of digital electronics with the University Guglielmo Marconi. His current research interests include reconfigurable computing, communication circuits, digital signal processing, and machine learning.

ROCCO FAZZOLARI received the master' degree in electronic engineering and the Ph.D. degree in space systems and technologies from the University of Rome Tor Vergata, Italy, in 2009 and 2013, respectively. He is currently a Postdoctoral Fellow and an Assistant Professor with the Department of Electronic Engineering, University of Rome Tor Vergata. He involves in hardware implementation of high-speed systems for digital signals processing, machine learning, array of wireless sensor networks, and systems for data analysis of acoustic emission (AE) sensors (based on ultrasonic waves).

DANIELE GIARDINO received the B.S. and M.S. degrees in electronic engineering from the University of Rome Tor Vergata, Italy, in 2015 and 2017, respectively, where he is currently pursuing the Ph.D. degree in electronic engineering. He is a member of the DSPVLSI Research Group, University of Rome Tor Vergata. He involves in digital development for wideband signals architectures, telecommunications, digital signal processing, and machine learning. In specific, he is focused on the digital implementation of MIMO systems for wideband signals.

ALBERTO NANNARELLI (S'94–M'99–SM'13) graduated in electrical engineering from the University of Roma La Sapienza, Roma, Italy, in 1988, and received the M.S. and the Ph.D. degrees in electrical and computer engineering from the University of California at Irvine, CA, USA, in 1995 and 1999, respectively. He was a Design Engineer with SGS-Thomson Microelectronics and Ericsson Telecom and a summer Intern with Rockwell Semiconductor Systems. From 1999 to 2003, he was with the Department of Electrical Engineering, University of Roma Tor Vergata, Italy, as a Postdoctoral Researcher. He is currently an Associate Professor with the Danmarks Tekniske Universitet, Lyngby, Denmark. His current research interests include computer arithmetic, computer architecture, and VLSI design. Dr. Nannarelli is a Senior Member of the IEEE Computer Society.

MARCO RE (M'92) received the Ph.D. degree in microelectronics from the University of Rome Tor Vergata, where he is currently an Associate Professor teaching digital electronics and hardware architectures for DSP. He received two NATO fellowships from the University of California at Berkeley while working as a Visiting Scientist with Cadence Berkeley Laboratories, and the Otto Monsted Fellowship when working as a Visiting Professor with the Technical University of Denmark. He collaborates in many research projects with different companies in the field of DSP architectures and algorithms. He is the author of about 200 articles on international journals and international conferences. His current research interests include low power DSP algorithms architectures, hardware-software codesign, fuzzy logic and neural hardware architectures, low power digital implementations based on non-traditional number systems, and computer arithmetic and cad tools for DSP. He is a member of the Audio Engineering Society (AES). He is a Director of a master in audio engineering with the Department of Electronic Engineering, University of Rome Tor Vergata.

SERGIO SPANÒ received the B.S. and M.S. degrees in electronic engineering from the University of Tor Vergata, Rome, Italy, in 2015 and 2018, respectively, where he is currently pursuing the Ph.D. degree in electronic engineering. He is a member of the DSPVLSI Research Group, University of Tor Vergata. His current research interests include digital signal processing, machine learning, telecommunications, and ASIC/FPGA hardware design. He had industrial experiences in the space and telecommunications field. His current research topics relate to machine learning hardware implementations for embedded and low-power systems.