2020 54th Asilomar Conference on Signals, Systems, and Computers | 978-0-7381-3126-9/20/$31.00 ©2020 IEEE | DOI: 10.1109/IEEECONF51394.2020.9443368

FPGA Implementation of Q-RTS for Real-Time Swarm Intelligence Systems

Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Matta, Alberto Nannarelli[(1)], Marco Re and Sergio Spanò

Department of Electronics, University of Rome Tor Vergata, Rome, Italy
g.cardarilli@uniroma2.it

[(1)]DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark

***Abstract*—We propose an architectural blueprint to implement Q-RTS, Q-Learning Real-Time Swarm Reinforcement Learning algorithm, on FPGA. The design solution is built on FPGA-based Centralized RL Processing Units (CRLPU). A CRLPU processes local and global state-action matrices and exchanges information frames with low-power Microcontroller-based Agents. The novel architecture implementation, for up to 32 Agents with up to 512 states, on a Xilinx Ultrascale device shows low resource requirements in terms of CLB (7%) and memory (2% FF and 22% BRAM). Performance metrics show that the required energy per generated action is always lower than 1 µJ.**

***Index Terms*—Machine Learning, Q-Learning, FPGA, Accelerator Architecture, Swarm Reinforement Learning, Robotics**

I. Introduction

In recent years, Reinforcement Learning (RL) field has become a trending research topic, in terms of Single-Agent RL (SARL), Multi-Agent RL (MARL) and Deep RL [1]–[5] algorithms, and digital hardware implementations [6]. In general, in RL, an entity called *Agent* learns to perform a task according to a trial-and-error process and infers an optimal *action-selection policy* [7] At a given time t, the quality of the performed action a_t when the Agent is in the state s_t is measured by a *reward value* r_t, i.e. the *reinforcement*. Q-Learning is a SARL algorithm [8] that aims for the optimal action-selection strategy by updating a table of Q-values which are performance figures of the Agent policy. In [9], authors proposed an optimized FPGA implementation for single Agent Q-Learning. In the field of MARL, we developed Q-RTS (*Q-Real Time Swarm for Intelligent systems* [10]). In this work we propose the architectural design approach for implementing different dedicated accelerator solutions, depending on the number of swarm Agents (2 to 32) and Agent states (32 to 512). In our experimental designs, the number of actions is 4, as they are sufficient to perform an optimization task in a 2D grid-world modeled environment, as reported in our previous research. The accelerator architecture is FPGA-based. The Centralized RL Processing Units (CRLPU) is the main core that implements the parallel Q-RTS algorithm.

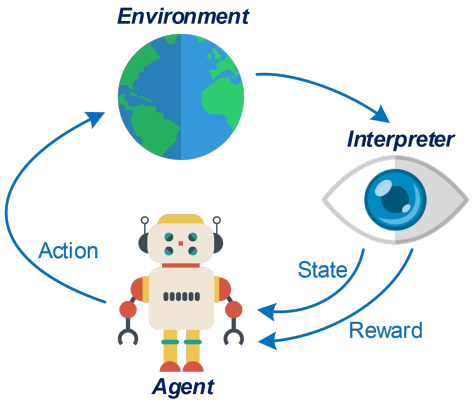

Fig. 1. Reinforcement Learning feedback principle

II. Machine Learning Background

A. Reinforcement Learning

ML techniques are usually included in three main categories: Supervised, Unsupervised and Reinforcement Learning. The first two expect a *training phase* to obtain an expert algorithm ready to be deployed in the field of application (*inference phase*). Massive amounts of example data and intensive offline training sessions often characterize Supervised and Unsupervised ML approaches. In Reinforcement Learning, training and inference are not split stages and take place at the same time directly on the application. As depicted in Fig. 1, the learner (*Agent*) interacts with the environment and, through the feedback of an *interpreter*, collects the state and an immediate reward which depends on the consequence of its action. The reward 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 the cumulative return by iterating actions over time. This process is modeled by the *State-Value Function* $V(s)$ which is updated using the reward value at each iteration [4]. By $V(s)$, the Agent learns the quality of the new state s_{t+1} reached from s_t by performing a_t. The process by which the Agent senses the environment and

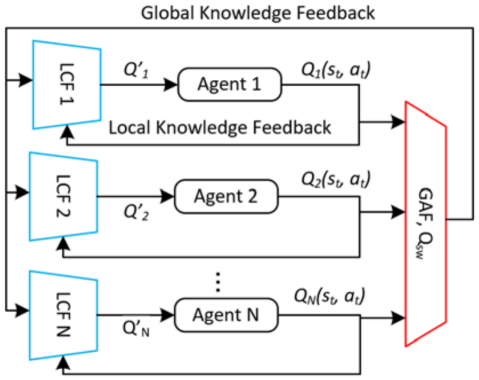

Fig. 2. Q-RTS double feedback

builds knowledge is called *exploration*. After the exploration, as $V(s)$ is updated, the Agent optimizes its set of actions in order to solve the task by maximizing the cumulative reward in fewer iterations. This second process is called *knowledge exploitation*. Both processes take place on the application field concurrently. Typical RL applications and contexts include:

- N-dimensional optimization problems, e.g. 2D grid world, mazes.
- Robotics, e.g. minimum path problems, walking robots.
- Autonomous vehicles, e.g. rovers, UAV.
- Disaster management e.g. unknown/hazardous space exploration.

B. Watkins Q-Learning (SARL)

Q-learning by Watkins is a Single-Agent Reinforcement Learning method capable to approximate the ideal State-Value function $V(s)$ of an Agent in order to find an optimal action-selection strategy. It's been demonstrated in [8] that it is sufficient to compute a map called *Action-Value Matrix* $Q_{\pi^*} : S \times A$ to estimate the State-Value function $V(s)$ related to the optimal policy $\pi(s, a)$. The elements of Q represent the quality of the related sub-optimal policy $\pi^*(s_t, a_t)$. The Q values are updated at every iteration using the Q-Learning update rule in (1):

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha\left(r_t + \gamma \max_a(Q(s_{t+1}, \cdot)\right) \quad (1)$$

where α is the learning rate, γ is the discount factor and $r_t$is the reward at time t. The Q-Learning Agent, in a state s_t, selects the action a_t by using the $Q(s_t, \cdot)$ row elements according one action-selection rule. The most important rules are:

- *Random*: the Agent ignores the Q-values and randomly selects a_t.
- *Greedy*: the action with the largest Q-value is selected.
- ϵ-*Greedy*: a mixed approach. The probability of Random action is ϵ, while the one of Greedy action is $1 - \epsilon$.

C. Q-RTS: Q-Real-Time Swarm Intelligence (MARL)

In RL algorithms for multi-agent systems the agents are usually organized in groups and show enhanced learning capabilities. Such approaches are often inspired by biological ensembles and swarms like ant or bee colonies [5]. Coordination, knowledge sharing and handling among the agents are key points in swarm-Multi Agent RL. Many MARL algorithms trigger the agent information exchange when all the agents reach a goal state (*episode*). Therefore, the update of an agent State-Value function is halted until the last agent reaches a terminal state. The authors in [10] presented a MARL algorithm that overcomes the bottleneck of the episodic knowledge handling by making it iterative: Q-Learning Real-Time Swarm algorithm (Q-RTS), a modified version of Q-Learning by Watkins. It is iteration-based and consequently suitable for real-time systems. Q-RTS is an algorithm based on a double feedback loop on an N-Agent swarm, as depicted in Fig. 2 :

- **Global Knowledge Feedback**: a Global Aggregation Function (GAF) combines the Q-matrices of all agents and delivers the global knowledge matrix (Q_{sw}).
- **Local Knowledge Feedback**: the Local Combination Functions (LCF) perform a weighted combination between the global knowledge matrix (Q_{sw}) and the single agents Q-matrices.

One of the key points of Q-RTS is the Q_{sw} matrix computation. All agents (GAF) contribute to the compilation of this matrix at each iteration: an agent Q-value corresponding to the largest reward (maximum value) or failure (minimum value) is assigned to the global knowledge Q_{sw} at the relevant state-action coordinate. This method, modeled as in (2) allows to quickly build the knowledge among all agents. Computing Q_{sw} is characterized by high parallelism that scales with the number of agents.

$$Q_{sw}(s_t, a_t) = \begin{cases} \max\limits_{Q_i \in \Pi} Q_i(s_t, a_t), \text{ if } \left|\max\limits_{Q_i \in \Pi} Q_i(s_t, a_t)\right| > \left|\min\limits_{Q_i \in \Pi} Q_i(s_t, a_t)\right| \\ \min\limits_{Q_i \in \Pi} Q_i(s_t, a_t), \text{ otherwise.} \end{cases} \quad (2)$$

where Π is the set of all the local agent matrices Q_i.

The second key point is the handling and distribution of knowledge. The Global Knowledge Feedback routes the Q_{sw} values to each agent node. The LCF in (3) performs a linear combination between the Agent Q-Matrix and the Swarm Q-matrix at each algorithm iteration. β is called *independence factor*, and weights the global component on the local matrix update.

$$Q'_i(s_t, a_t) = \beta Q_i(s_t, a_t) + (1-\beta)Q_{sw}(s_t, a_t) \quad (3)$$

At this section, the Local Agent Q-matrix update takes place, which follows a modified Q-Learning rule (4). It updates Q_i

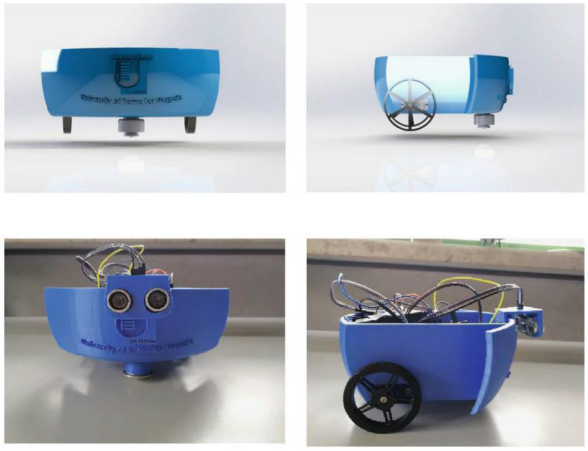

Fig. 3. Q-RTS Agent rover prototype

using the weighted Q_i' matrix instead of Q_i itself (Local Knowledge Feedback).

$$Q_i(s_t, a_t) \leftarrow (1-\alpha)Q_i'(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q_i'(s_{t+1}, \cdot)\right) \quad (4)$$

The ultimate purpose of the LCF is to provide the agent a β-degree of independence in building the knowledge locally: Q_{sw} might contain outdated information if the environment changes over time. The set of LCF is an array of linear interpolators. Their total computational complexity scales with the number of agents, as in the GAF case.

III. Q-RTS FPGA Accelerator

A. Algorithm breakdown

The Q-RTS algorithm is characterized by several computation nodes that update the memory elements at each iteration. In particular, the following operations are required for each Agent:

- **Single agent Q-Matrix Q_i' update**, according (4). In particular, the Q-Table update value is calculated using:
  - *3 products*: 2 to α and one to γ, that are taken as constant values for the sake of simplicity.
  - *2 sums*: r_t plus the learned value $\max_a Q_i'(s_{t+1}, \cdot)$ and the old Q-value added to the first sum.
  - *1 Maximum search*: at each iteration, the next state action with the maximum Q-value. Considering N_a available actions, the complexity scales with $\log_2 N_a$.
- The **Local Combination Functions** (3) are linear interpolations between $Q_{sw}(s_t, a_t)$ and the $Q(s_t, a_t)$ elements, weighted by the β independence constant parameter.
- The **The Global Aggregation Function** (2) computation directly affects the N-Agent Swarm. At each iteration, the global knowledge matrix Q_{sw} is filled after:
  - $2N$ absolute values.
  - Maximum search among N elements ($log_2 N$ complexity).

Taking into consideration that the Single Agent Matrix update and the Local Combination Function are used for each Agent in the swarm, the overall complexity scales with N. However, they represent separated tasks that can be executed independently in parallel. Similarly, the Global Aggregation Function depends on the computation of N-parallel absolute value and maximum search tasks that can be executed concurrently. The total memory accesses ($Q_{sw}(s_t, a_t)$, $Q_i'(s_t, a_t)$ and $Q_i(s_t, a_t)$) is dependent on the number of states, actions and Agents.

B. Centralized Processing Reinforcement Learning Unit

An accelerator architecture design for Q-RTS depends on the field of application. A swarm of autonomous agent-vehicles (rovers in Fig. 3) is the case study which the architectural choices are based on. The number of actions is 4 (*move forward, move backwards, rotate clockwise, rotate counterclockwise*), sufficient to explore a bi-dimensional space, and different number of states, depending on the design solution. The agent sensors and actuators are equipped on the rover itself, while the single agent update is performed on a different platform. The Q-RTS accelerator, called *Centralized RL Processing Units* (CRLPU), is the main hub and part of a star-topology network system (Fig. III-A) which features:

- A FPGA that implements and accelerates all the parallel computation computation nodes described in the previous section on FPGA.
- A number of decentralized Agent Sensors and Actuators that communicate with the CRLPU by a wireless link. States are computed on-board and sent to the CRLPU FPGA, while actions are delivered to all agents from the central star node.

The CRLPU-FPGA architecture is shown in Figure 5. It includes the following:

1) Agents Update Array: The agent Q-table is updated on dedicated modules on FPGA. The single module architecture, in Fig. 6, is based on the one presented in [9]. Green inputs and outputs are the state value read remotely on-board and the action command that is sent to the rover. Red labeled inputs and outputs are connected to the two feedback networks of Q-RTS (*Local Knowledge* and *Global Knowledge Feedback*).

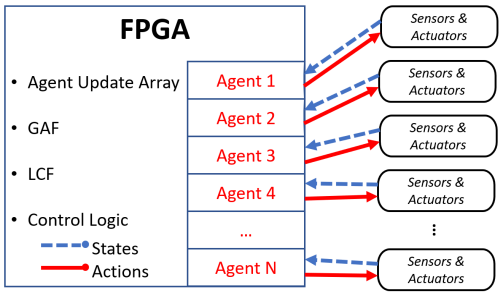

Fig. 4. Centralized Reinforcement Learning Processing Unit and decentralized Agent Sensors/Acturators

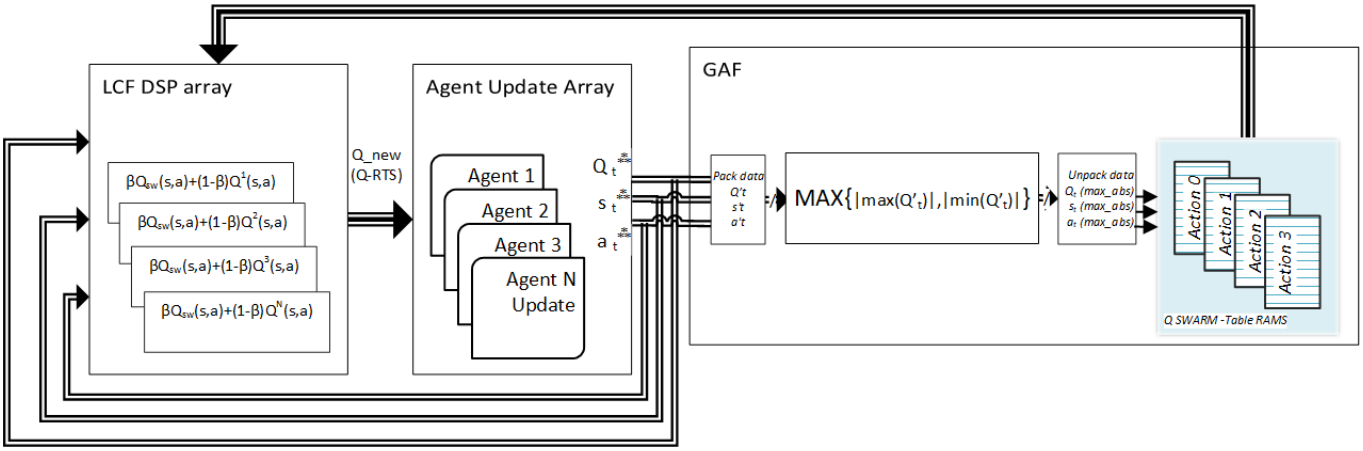

Fig. 5. Centralized Reinforcement Learning Processing Unit

The Agent Q-Table is stored in 4 RAMs, one for each action, having a number of elements equal to the number of states. The stored Q-values are a 12-bit words. An action decoder is used to enable the read and write operations on the RAM relevant to the current action a_t and next a_{t+1} actions respectively. The *Action-Selection Policy* block implements one of the three main action-selection rules explained in Sect. II-B. In particular, a 32-bit Linear Feedback Shift Register is used to compute both the random action number and the ϵ-Greedy event. The action-selection mode is programmable by rule type and ϵ probability. The Q-Update block is the processing core of this module. It implements (4), both α and γ are programmable and the maximum search between the four $Q(s_{t+1}, \cdot)$ values is performed by a 2-level comparator tree. Delays d are used to adjust the internal feedback timing between the current (t) and the next $(t+1)$ state-action pairs. A multiplexer for data-path selection (*LCF-Q selector*) is used to switch the Agent mode from standard Q-Learning to Q-RTS. This is the main difference in the architecture compared to [9].

2) Global Aggregation Function module: The Aggregation Function accelerator accepts the $\{Q(s,a), s, a\}$ 3-uples from all agents as inputs. The data frames are compared in terms of absolute values and the maximum Q_i^{max} is stored in the Q_{sw} table in the relevant (s_i^{max}, a_i^{max}) coordinates. The maximum search is performed by means of parallel N 2-level comparators. The Q_{sw} table is implemented in 4 RAM memories, reflecting the single agent accelerator architecture.

3) Local Combination Function DSP array: The double feedback data is routed from the Agent Update Array and the Global Aggregation Function module to the LCF DSP array. The tabular data is used to compute the linear interpolation (3) equations and a pair of multiply-sum blocks is employed for each agent. The results are delivered to the Agent Update modules to be used in the Q-RTS internal data-path.

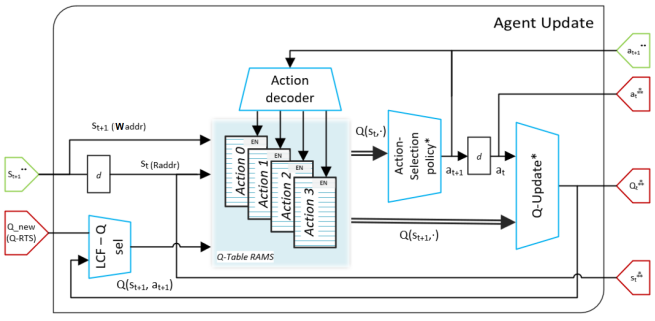

Fig. 6. Q-RTS Agent Update Module

IV. Implementation Method, Results and Performance

Different design solutions, depending on the number of swarm agents (2 to 32) and agent states (32 to 512) were implemented on a Xilinx xczu7ev-ffvc1156 device (Ultrascale). The architecture was described using an HLS approach. Q-RTS, as an algorithm, fits the high-level description method:

- The i-th swarm agent is implemented as an object of the custom Agent class.
- The Q-RTS feedback loops feature high degree of parallelism that can be easily set by using the dedicated HLS directives.

The HDL synthesizer was set to obtain the best effort trade-off between occupied resources and execution latency of the Q-RTS algorithm. As reported in Table I, the CRLPU design solutions require a small amount of combinational logic and memory elements, relatively to the device. A maximum of 2% of CLBs, 7.1% of LUTs and 1.9% of FFs in the case of 512-state 32-agent Q-RTS are required. The use of embedded memory blocks (BRAM) is below the 21% of the total available. BRAMs have not been used for Q-RTS agents with 32 states. The number of DSP primitives grows with the number of agents in the swarm, especially due to the larger combinational complexity of the LCF blocks. Regarding performance, latency increases by a factor of 191 due to the computational load of LCF array. However, the throughput, considered in terms of millions of actions per second (MAps), reduces only by

TABLE I
Q-RTS CRLPU IMPLEMENTATION RESULTS

Q-RTS Type		Resource Utilization (quantity, % of total)					Performance Table				
N	States	CLB	LUT	FF	DSP	BRAM	P_d (mW)	Max fclk (MHz)	Latency (μ s)	MApS	EpA (nJ)
2	32	176 (0.1%)	779 (0.3%)	837 (0.2%)	11 (0.6%)	0 (0.0%)	11	228.47	0.10	20.77	0.53
2	128	138 (0.1%)	647 (0.3%)	581 (0.1%)	11 (0.6%)	12 (1.9%)	7	190.01	0.12	17.27	0.41
2	512	139 (0.1%)	656 (0.3%)	622 (0.1%)	11 (0.6%)	12 (1.9%)	7	188.86	0.12	17.17	0.41
4	32	316 (0.1%)	1554 (0.7%)	1548 (0.3%)	31 (1.8%)	0 (0.0%)	33	156.03	0.32	12.48	2.64
4	128	273 (0.1%)	1318 (0.6%)	1138 (0.3%)	31 (1.8%)	20 (3.2%)	14	137.89	0.36	11.03	1.27
4	512	305 (0.1%)	1358 (0.6%)	1208 (0.3%)	31 (1.8%)	20 (3.2%)	14	153.42	0.33	12.27	1.14
8	32	674 (0.3%)	3033 (1.3%)	2953 (0.6%)	91 (5.3%)	0 (0.0%)	111	158.96	0.97	8.26	13.44
8	128	586 (0.3%)	2625 (1.1%)	2231 (0.5%)	91 (5.3%)	36 (5.8%)	30	131.58	1.17	6.84	4.39
8	512	637 (0.3%)	2712 (1.2%)	2373 (0.5%)	91 (5.3%)	36 (5.8%)	32	131.13	1.17	6.81	4.70
16	32	1607 (0.7%)	6892 (3.0%)	5388 (1.2%)	309 (17.9%)	0 (0.0%)	329	135.81	4.08	3.92	83.88
16	128	1503 (0.7%)	6087 (2.6%)	4036 (0.9%)	309 (17.9%)	0 (0.0%)	373	120.6	4.59	3.48	107.09
16	512	1622 (0.7%)	6320 (2.7%)	4324 (0.9%)	309 (17.9%)	68 (10.9%)	72	116.13	4.77	3.35	21.47
32	32	4888 (2.1%)	17529 (7.6%)	10917 (2.4%)	1121 (64.9%)	0 (0.0%)	1238	110.07	19.28	1.66	745.83
32	128	4640 (2.0%)	15712 (6.8%)	8338 (1.8%)	1121 (64.9%)	132 (21.2%)	198	110.08	19.28	1.66	119.27
32	512	4695 (2.0%)	16423 (7.1%)	8941 (1.9%)	1107 (64.1%)	132 (21.2%)	1538	111.19	19.08	1.68	917.29

a factor 15 thanks to the high level of parallelism. Dynamic power (P_d) and the energy required to deliver an action (EpA) is low in any case, spanning from fractions of nJ to 0.917μJ. However, it significantly decreases when using BRAMs.

CONCLUSIONS AND FINAL REMARKS

We presented a first core architecture that implements the new Q-RTS algorithm and we provided the first benchmark data regarding implementation resources and performance a Xilinx UltraScale device.

This project has been developed in the frame of TORVE-BOT research project [11] by "Tor Vergata" University, Rome (dept. of Electronics Engineering and dept. of Insdustrial Engineering), in collaboration with DTU, Copenaghen. It aims to create a swarm of intelligent autonomous rovers that learn how to solve a given task. Through Reinforcement Learning and Q-RTS, the swarm of robots is capable of learning how to solve cooperatively a task by gathering information from the environment.

REFERENCES

[1] Capizzi, G., Lo Sciuto, G., Napoli, C., Polap, D., Wozniak, M. "Small Lung Nodules Detection Based on Fuzzy-Logic and Probabilistic Neural Network with Bioinspired Reinforcement Learning". (2020) IEEE Transactions on Fuzzy Systems, 28 (6), art. no. 8895990, pp. 1178-1189.

[2] Matta, M., Cardarilli, G.C., Di Nunzio, L., Fazzolari, R., Giardino, D., Nannarelli, A., Re, M., Spanò, S. "A reinforcement learning-based QAM/PSK symbol synchronizer". (2019) IEEE Access, 7, pp. 124147-124157.

[3] Botvinick, M., Ritter, S., Wang, J. X., Kurth-Nelson, Z., Blundell, C., & Hassabis, D. (2019). "Reinforcement learning, fast and slow. Trends in cognitive sciences".

[4] Drugan, M. M. (2019). "Reinforcement Learning versus evolutionary computation: A survey on hybrid algorithms". Swarm and evolutionary computation, 44, 228-246.

[5] Hüttenrauch, M., Adrian, S., & Neumann, G. (2019)." Deep reinforcement learning for swarm systems". Journal of Machine Learning Research, 20(54), 1-31.

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

[7] M. Mohri et al., "Introduction", in Foundations of Machine Learning, MIT press, 2018, pp. 6-7

[8] C. J. Watkins et al., "Q-Learning", Machine learning, vol. 8, n. 3-4, 1992, pp. 279-292.

[9] Spanò, S., Cardarilli, G. C., Di Nunzio, L., Fazzolari, R., Giardino, D., Matta, M., ... & Re, M. (2019). "An Efficient Hardware Implementation of Reinforcement Learning: The Q-Learning Algorithm". IEEE Access, 7, 186340-186351.

[10] Matta, M., Cardarilli, G. C., Di Nunzio, L., Fazzolari, R., Giardino, D., Re, M., ...& Spanò, S. (2019). "Q-RTS: a real-time swarm intelligence based on multi-Agent Q-learning". Electronics Letters, 55(10), 589-591.

[11] Di Nunzio, L., Cardarilli, G., Ceccarelli, M., Fazzolari, R., Design and Requirements for a Mobile Robot for Team Cooperation (2020) Mechanisms and Machine Science, 78, pp. 277-285.