Artificial Intelligence 256 (2018) 130–159

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Decentralized Reinforcement Learning of Robot Behaviors

David L. Leottau [a,*], Javier Ruiz-del-Solar [a], Robert Babuška [b,c]

[a] Department of Electrical Engineering, Advanced Mining Technology Center, Universidad de Chile, Av. Tupper 2007, Santiago, Chile
[b] Cognitive Robotics Department, Faculty of 3mE, Delft University of Technology, 2628 CD Delft, The Netherlands
[c] CIIRC, Czech Technical University in Prague, Czech Republic

ARTICLE INFO

Article history:
Received 3 April 2017
Received in revised form 10 November 2017
Accepted 4 December 2017
Available online 11 December 2017

Keywords:
Reinforcement learning
Multi-agent systems
Decentralized control
Autonomous robots
Distributed artificial intelligence

ABSTRACT

A multi-agent methodology is proposed for Decentralized Reinforcement Learning (DRL) of individual behaviors in problems where multi-dimensional action spaces are involved. When using this methodology, sub-tasks are learned in parallel by individual agents working toward a common goal. In addition to proposing this methodology, three specific multi agent DRL approaches are considered: DRL-Independent, DRL Cooperative-Adaptive (CA), and DRL-Lenient. These approaches are validated and analyzed with an extensive empirical study using four different problems: 3D Mountain Car, SCARA Real-Time Trajectory Generation, Ball-Dribbling in humanoid soccer robotics, and Ball-Pushing using differential drive robots. The experimental validation provides evidence that DRL implementations show better performances and faster learning times than their centralized counterparts, while using less computational resources. DRL-Lenient and DRL-CA algorithms achieve the best final performances for the four tested problems, outperforming their DRL-Independent counterparts. Furthermore, the benefits of the DRL-Lenient and DRL-CA are more noticeable when the problem complexity increases and the centralized scheme becomes intractable given the available computational resources and training time.

1. Introduction

Reinforcement Learning (RL) is commonly used in robotics to learn complex behaviors [43]. However, many real-world applications feature multi-dimensional action spaces, i.e. multiple actuators or effectors, through which the individual actions work together to make the robot perform a desired task. Examples are multi-link robotic manipulators [7,31], mobile robots [11,26], aerial vehicles [2,16], multi-legged robots [47], and snake robots [45]. In such applications, RL suffers from the combinatorial explosion of complexity, which occurs when a Centralized RL (CRL) scheme is used [31]. This leads to problems in terms of memory requirements or learning time and the use of *Decentralized Reinforcement Learning* (DRL) helps to alleviate these problems. In this article, we will use the term DRL for decentralized approaches to the learning of a task which is performed by a single robot.

In DRL, the learning problem is decomposed into several sub-problems, whose resources are managed separately, while working toward a common goal. In the case of multidimensional action spaces, a sub-task corresponds to controlling one particular variable. For instance, in mobile robotics, a common high-level motion command is the desired velocity vector (e.g., $[v_x, v_y, v_\theta]$), and in the case of a robotic arm, it can be the joint angle setpoint (e.g., $[\theta_{shoulder}, \theta_{elbow}, \theta_{wrist}]$). If each

* Corresponding author.
E-mail address: dleottau@ing.uchile.cl (D.L. Leottau).

https://doi.org/10.1016/j.artint.2017.12.001
0004-3702/

component of this vector is controlled individually, a distributed control scheme can be applied. Through coordination of the individual learning agents, it is possible to use decentralized methods [7], taking advantage of parallel computation and other benefits of Multi-Agent Systems (MAS) [42,5].

In this work, a Multi-Agent (MA) methodology is proposed for modeling the DRL in problems where multi-dimensional action spaces are involved. Each sub-task (e.g., actions of one effector or actuator) is learned by a separate agent and the agents work in parallel on the task. Since most of the MAS reported studies do not validate the proposed approaches with multi-state, stochastic, and real world problems, our goal is to show empirically that the benefits of MAS are also applicable to complex problems like robotic platforms, by using DRL systems. In this paper, three Multi-Agent Learning (MAL) algorithms are considered and tested: the independent DRL, the Cooperative Adaptive (CA) Learning Rate, and a Lenient learning approach extended to multi-state DRL problems.

The independent DRL (DRL-Ind) does not consider any kind of cooperation or coordination among the agents, applying single-agent RL methods to the MA task. The Cooperative Adaptive Learning Rate DRL (DRL-CA) and the extended Lenient DRL (DRL-Lenient) algorithms add coordination mechanisms to the independent DRL scheme. These two MAL algorithms are able to improve the performance of those DRL systems in which complex scenarios with several agents with different models or limited state space observability appear. Lenient RL was originally proposed by Panait, Sullivan, and Luke [35] for stateless MA games; we have adopted it to multi-state and stochastic DRL problems based on the work of Schuitema [39]. On the other hand, the DRL-CA algorithm uses similar ideas to those of Bowling and Veloso [3], and Kaisers and Tuyls [19] for having a variable learning rate, but we are introducing direct cooperation between agents without using joint actions information and not increasing the memory consumption or the state space dimension.

The proposed DRL methodology and the three MAL algorithms considered are validated through an extensive empirical study. For that purpose, four different problems are modeled, implemented, and tested; two of them are well-known problems: an extended version of the Three-Dimensional Mountain Car (3DMC) [46], and a SCARA Real-Time Trajectory Generation (SCARA-RTG) [31]; and two correspond to noisy and stochastic real-world mobile robot problems: Ball-Dribbling in soccer performed with an omnidirectional biped robot [25], and the Ball-Pushing behavior performed with a differential drive robot [28].

In summary, the main contributions of this article are threefold. First, we propose a methodology to model and implement a DRL system. Second, three MAL approaches are detailed and implemented, two of them including coordination mechanisms. These approaches, DRL-Ind, DRL-CA, and DRL-Lenient, are evaluated on the above-mentioned four problems, and conclusions about their strengths and weaknesses are drawn according to each validation problem and their characteristics. Third, to the best of our knowledge, our work is the first one that applies a decentralized modeling to the learning of individual behaviors on mobile robot platforms, and compares it with a centralized RL scheme. Further, we expect that our proposed extension of the 3DMC can be used in future work as a test-bed for DRL and multi-state MAL problems. Finally, all the source codes are shared at our code repository [23], including our custom hill-climbing algorithm for optimizing RL parameters.

This remainder of this paper is organized as follows: Section 2 presents the preliminaries and an overview of related work. In Section 3, we propose a methodology for modeling DRL systems, and in Section 4, three MA based DRL algorithms are detailed. In Section 5, validation problems are specified and the experiments, results, and discussion are presented. Finally, Section 6 concludes the paper.

2. Preliminaries

This section introduces the main concepts and background based on the works of Sutton and Barto [43], and Busoniu, Babuska, De-Schutter, and Ernst [6] for single-agent RL; Busoniu, Babuska, and De Schutter [5], and Vlassis [52] for Multi-Agent RL (MARL); Laurent, Matignon, and Fort-Piat [22] for independent learning; and Busoniu, De-Schutter, and Babuska [7] for decentralized RL. Additionally, an overview of related work is presented.

2.1. Single-agent reinforcement learning

RL is a family of machine learning techniques in which an agent learns a task by directly interacting with the environment. In the single-agent RL, studied in the remainder of this article, the environment of the agent is described by a Markov Decision Process (MDP), which considers stochastic state transitions, discrete time steps $k \in \mathbb{N}$ and a finite sampling period.

Definition 1. A finite *Markov decision process* is a 4-tuple $\langle S, A, T, R \rangle$ where: S is a finite set of environment states, A is a finite set of agent actions, $T : S \times A \times S \to [0, 1]$ is the state transition probability function, and $R : S \times A \times S \to \mathbb{R}$ is the reward function [5].

The stochastic state transition function T models the environment. The state of the environment at discrete time-step k is denoted by $s_k \in S$. At each time step, the agent can take an action $a_k \in A$. As a result of that action, the environment changes its state from s_k to s_{k+1}, according $T(s_k, a_k, s_{k+1})$, which is the probability of ending up in s_{k+1} given that action a_k is applied in s_k. As an immediate feedback on its performance, the agent receives a scalar reward $r_{k+1} \in R$, according to

the reward function: $r_{k+1} = R(s_k, a_k, s_{k+1})$. The behavior of the agent is described by its policy π, which specifies how the agent chooses its actions given the state.

This work is about tasks that require several simultaneous actions (e.g., a robot with multiple actuators), where such tasks are learned by using separate agents, one for each action. In this setting, the state transition probability depends on the actions taken by all the individual agents. We consider on-line and model-free algorithms, as they are convenient for practical implementations.

Q-Learning [53] is one of the most popular model-free, on-line learning algorithms. It turns Bellman equation into an iterative approximation procedure which updates the Q-function by the following rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)\right] \tag{1}$$

in which s' and a' correspond to s_{k+1} and a_{k+1} respectively, with $\alpha \in (0,1]$ the learning rate, and $\gamma \in (0,1)$ the discount factor. The sequence of Q-functions provably converges to Q^* under certain conditions, including that the agent keeps trying all actions in all states with non-zero probability.

2.2. Multi-agent learning

The generalization of the MDP to the multi-agent case is the stochastic game.

Definition 2. A stochastic game is the tuple $\langle S, A^1, \cdots, A^M, T, R^1 \ldots R^M \rangle$ with M the number of agents; S the discrete set of environment states; $A^m, m = 1, \cdots, M$ the discrete sets of actions available to the agents, yielding the joint action set $\mathcal{A} = A^1 \times \cdots \times A^M$; $T: S \times \mathcal{A} \times S \to [0,1]$ the state transition probability function, such that, $\forall s \in S, \forall a \in \mathcal{A}, \sum_{s_{k+1} \in S} T(s_k, a_k, s_{k+1}) = 1$; and $R^m: S \times \mathcal{A} \times S \to \mathbb{R}, m = 1, \cdots, M$ the reward functions of the agents [5,22].

In the multi-agent case, the state transitions depend on the joint action of all the agents, $a_k = [a_k^1, \cdots, a_k^M], a_k \in \mathcal{A}, a_k^m \in A^m$. Each agent may receive a different reward r_{k+1}^m. The policies $\pi^m : S \times A^m \to [0,1]$ form together the joint policy π. The Q-function of each agent depends on the joint action and is conditioned on the joint policy, $Q_m^\pi: S \times \mathcal{A} \to \mathbb{R}$.

If $R^1 = \cdots = R^M$, all the agents have the same goal, and the stochastic game is fully cooperative. If $M = 2$, $R^1 = -R^2$, and they sum-up to zero, the two agents have opposite goals, and the game is fully competitive. Mixed games are stochastic games that are neither fully cooperative nor fully competitive [52]. In the general case, the reward functions of the agents may differ. Formulating a good learning goal in situations where the agents' immediate interests are in conflict is a difficult open problem [7].

2.2.1. Independent learning

Claus and Boutilier [8] define two fundamental classes of agents: joint-action learners and Independent Learners (ILs). Joint-action learners are able to observe the other agents actions and rewards; those learners are easily generalized from standard single-agent RL algorithms as the process stays Markovian. On the contrary, ILs do not observe the rewards and actions of the other learners, they interact with the environment as if no other agents exist [22].

Most MA problems violate the Markov property and are non-stationary. A process is said non-stationary if its transition probabilities change with the time. A non-stationary process can be Markovian if the evolution of their transition and reward functions depends only on the time step and not on the history of actions and states [22].

For ILs, which is the focus of the present paper, the individual policies change as the learning progresses. Therefore, the environment is non-stationary and non-Markovian. Laurent, Matignon and Fort-Piat [22] give an overview of strategies for mitigating convergence issues in such a case. The effects of agents' non-stationarity are less observable in weakly coupled distributed systems, which makes ILs more likely to converge. The observability of the actions' effects may influence the convergence of the algorithms. To ensure convergence, these approaches require the exploration rate to decay as the learning progresses, in order to avoid too much concurrent exploration. In this way, each agent learns the best response to the behavior of the others. Another alternative is to use coordinated exploration techniques that exclude one or more actions from the agent's action space, to efficiently search in a shrinking joint action space. Both approaches reduce the exploration, the agents evolve slower and the non-Markovian effects are reduced [22].

2.3. Decentralized Reinforcement Learning

DRL is concerned with MAS and Distributed Problem Solving [42]. In DRL, a problem is decomposed into several subproblems, managing their individual information and resources in parallel and separately, by a collection of several agents which are part of a single entity, such as a robot. For instance, consider a quadcopter learning to perform a maneuver: each rotor can be considered as a subproblem rather than an entirely independent problem; each subproblem's information and resources (sensors, actuators, effectors, etc.) can be managed separately by four agents; so, four individual policies will be learned to perform the maneuver in a collaborative way.

One of the first mentions of DRL is by Busoniu, De-Schutter, and Babuska [7], where it was used to differentiate a decentralized system from a MAL system composed of individual agents [34]. The basic DRL architecture is shown in Fig. 1

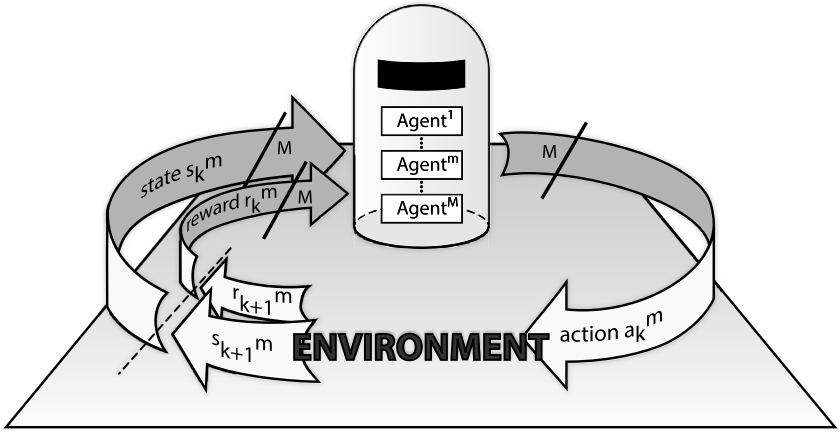

Fig. 1. The basic DRL architecture.

where M individual agents are interacting within an environment. According to Tuyls, Hoen, and Vanschoenwinkel [49], single-agents working on a multi-agent task are able to converge to a coordinate equilibrium under certain parameters and for some particular behaviors. In this work we validate that assumption empirically with several problems in which multi-dimensional action spaces are present. Thus, a methodology for modeling those problems by using a DRL system is a primary contribution of this work.

2.3.1. Potential advantages of DRL

One of the main drawbacks of classical RL is the exponential increase of complexity with the number of state variables. Moreover, problems with multi-dimensional action spaces suffer from the same drawback in the action space, too. This makes the learning process highly complex, or even intractable, in terms of memory requirements or learning time [31]. This problem can be overcome by addressing it from a DLR perspective. For instance, by considering a system with M actuators (an M-dimensional action space) and N discrete actions in each one, a DRL modeling leads to evaluating and storing NM values per state instead of N^M as a centralized RL does. This result in a linear increase with the number of actuators instead of an exponential one. A generalized expression for memory requirements and a computation time reduction factor during action selection can be determined [39], this is one of the main benefits of using DRL over CRL schemes, expressed by the following ratio:

$$\frac{\prod_{m=1}^{M} |N^m|}{\sum_{m=1}^{M} |N^m|}, \tag{2}$$

where actuator m has $|N^m|$ discrete actions.

The MAS perspective grants several potential advantages if the problem is approached with decentralized learners:

- Since from a computational point of view, all the individual agents in a DRL system can operate in parallel acting upon their individual and reduced action spaces, the learning speed is typically higher compared to a centralized agent which searches an exponentially larger action space $N = N^1 \times \cdots \times N^M$, as expressed in (2) [39].
- The state space can be reduced for an individual agent, if not all the state information is relevant to that agent.
- Different algorithms, models or configurations could be used by each individual agent.
- Parallel or distributed computing implementations are suitable.

There are various alternatives to decentralize a system performed with a single robot, for example, task decomposition [54], behavior fusion [17], and layered learning [44]. However, in this work we are proposing the multi-dimensional action space decomposition, where each action dimension is learned-controlled by one agent. In this way, the aforementioned potential advantages can be exploited.

2.3.2. Challenges in DRL

DRL also has several challenges which must be resolved efficiently in order to take advantage of the benefits already mentioned. Agents have to coordinate their individual behaviors toward a desired joint behavior. This is not a trivial goal since the individual behaviors are correlated and each individual decision influences the environment. Furthermore, as pointed out in Section 2.2.1, an important aspect to deal with is the Markov property violation. The presence of multiple concurrent learners, makes the environment non-stationary from a single agent's perspective [5]. The evolution of its transition probabilities do not only depend on time, the process evolution is led by the agents' actions and their own history. Therefore,

from a single agent's perspective, the environment no longer appears Markovian [22]. In Section 4, two MAL algorithms for addressing some of these open issues in DRL implementations, are presented: the Cooperative Adaptive Learning Rate, and an extension of the Lenient RL algorithm applied to multi-state DRL problems.

2.4. Related work

Busoniu et al. [7] present centralized and multi-agent learning approaches for RL, tested on a two-link manipulator, and compared them in terms of performance, convergence time, and computational resources. Martin and De Lope [31] present a distributed RL modeling for generating a real-time trajectory of both a three-link planar robot and the SCARA robot; experimental results showed that it is not necessary for decentralized agents to perceive the whole state space in order to learn a good global policy. Probably, the most similar work to ours is reported by Troost, Schuitema, and Jonker [48], this paper uses an approach in which each output is controlled by an independent $Q(\lambda)$ agent. Both simulated robotic systems tested showed and almost identical performance and learning time between the single-agent and MA approaches, while this last one requires less memory and computation time. A Lenient RL implementation was also tested showing a significant performance improvement for one of the case studied. Some of these experiments and results were extended and presented by Schuitema [39]. Moreover, the DRL of the soccer ball-dribbling behavior is accelerated by using knowledge transfer [26], where, each component of the omnidirectional biped walk (v_x, v_y, v_θ) is learned by separate agents working in parallel on a multi-agent task. This learning approach for the omnidirectional velocity vector is also reported by Leottau, Ruiz-del-Solar, MacAlpine, and Stone [27], in which some layered learning strategies are studied for developing individual behaviors, and one of these strategies, the concurrent layered learning involves the DRL. Similarly, a MARL application for the multi-wheel control of a mobile robot is presented by Dziomin, Kabysh, Golovko, and Stetter [11]. The robotic platform is separated into driving module agents that are trained independently, in order to provide energy consumption optimization. A multi-agent RL approach is presented by Kabysh, Golovko, and Lipnickas [18], which uses agents' influences to estimate learning error among all the agents; it has been validated with a multi-joint robotic arm. On the other hand, Kimura [20] presents a coarse coding technique and an action selection scheme for RL in multi-dimensional and continuous state-action spaces following conventional and sound RL manners; and Pazis and Lagoudakis [38] present an approach for efficiently learning and acting in domains with continuous and/or multidimensional control variables, in which the problem of generalizing among actions is transformed to a problem of generalizing among states in an equivalent MDP, where action selection is trivial. A different application is reported by Matignon, Laurent, and Fort-Piat [32], where a semi-decentralized RL control approach for controlling a distributed-air-jet micro-manipulator is proposed. This showed a successful application of decentralized Q-learning variant algorithms for independent agents. Finally, a well-known related work was reported by Crites and Barto [9], an application of RL to the real world problem of elevator dispatching, its states are not fully observable and they are non-stationary due to changing passenger arrival rates. So, a team of RL agents is used, each of which is responsible for controlling one elevator car. Results showed that in simulation surpass the best of the heuristic elevator control tested algorithms.

3. Decentralized Reinforcement Learning methodology

In this section, we present a methodology for modeling and implementing a DRL system. Aspects such as what kind of problem is a candidate for being decentralized, which sub-tasks, actions, or states should or could be decomposed, and what kind of reward functions and RL learning algorithms should be used are addressed. The 3D mountain car is used as an example in this section. The following methodology is proposed for identifying and modeling a DRL system.

3.1. Determining if the problem is decentralizable

First of all, it is necessary to determine if the problem addressed is decentralizable via action space decomposition, and, if it is, to determine into how many subproblems the system can be separated. In robotics, a multi-dimensional action space usually implies multiple controllable inputs, i.e., multiple actuators or effectors. For instance, an M-joint robotic arm or an M-copter usually has at least one actuator (e.g., a motor) per joint or rotor, respectively, while a differential drive robot has two end-effectors (right and left wheels), and an omnidirectional biped gait has a three-dimensional commanded velocity vector ($[v_x, v_y, v_\theta]$). Thus, for the remainder of this approach, we are going to assume as a first step that:

Proposition 1. *A system with an M-dimensional action space is decentralizable if each of these action dimensions are able to operate in parallel and their individual information and resources can be managed separately. In this way, it is possible to decentralize the problem by using M individual agents, which will learn together toward a common goal.*

This concept will be illustrated with the 3DMC problem. A basic description of this problem is given below, and it will be detailed in depth in Section 5.1.

3-Dimensional mountain car: mountain car is one of the canonical RL tasks where an agent must drive an under-powered car up a mountain to reach a goal state. In the 3D modification originally proposed by Taylor and Stone [46], the mountain's curve is extended to a 3D surface as is shown in Fig. 2. The state has four continuous state variables: $[x, \dot{x}, y, \dot{y}]$. The agent

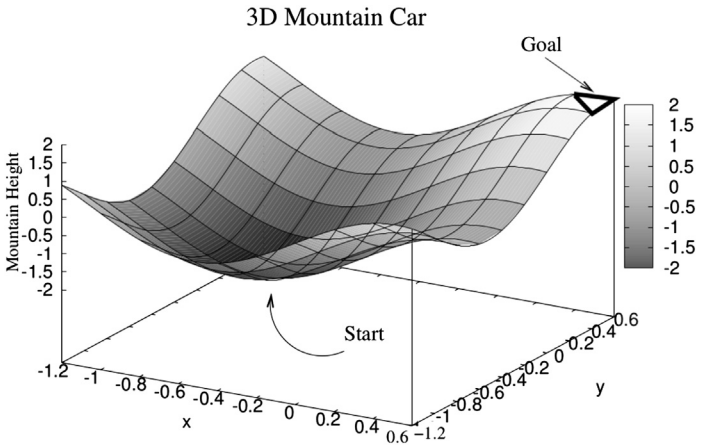

Fig. 2. 3D mountain car surface. Figure adopted from Taylor and Stone [46].

selects from five actions: $\{Neutral, West, East, South, North\}$, where the x axis of the car points toward the north. The reward is -1 for each time step until the goal is reached, at which point the episode ends and the reward is 0.

This problem can also be described by a decentralized RL modeling. It has a bi-dimensional action space, where $\{West, East\}$ actions modify speed $\dot{x}$ onto the x axis (dimension 1), and $\{South, North\}$ actions modify speed $\dot{y}$ onto the y axis (dimension 2). These two action dimensions can act in parallel, and they can be controlled separately. So, Proposition 1 is fulfilled, and 3DMC is a decentralizable problem by using two RL separate agents: $Agent_x$ and $Agent_y$.

3.2. Identifying common and individual goals

In a DRL system, a collection of separate agents learn, individual policies in parallel, in order to perform a desired behavior together to reach a *common goal*.

A common goal is always present in a DRL problem, and for some cases it is the same for all the individual agents, especially when they share identical state spaces, similar actions, and common reward functions. But, there are problems in which a particular sub-problem can be assigned to a determined agent in order to reach that common goal. To identify each agent's individual goal is a non-trivial design step, that requires knowledge of the problem. This is not an issue for centralized schemes, but it is an advantage of a decentralized modeling because it allows addressing the problem more deeply.

There are two types of individual goals for DRL agents: those which are intrinsically generated by the learning system when an agent has different state or action spaces with respect to the others, and those individual goals which are assigned by the designer to every agent, defining individual reward functions for that purpose. For the remainder of this manuscript, the concept of individual goals and individual reward functions will refer to those kinds of goals assigned by the designer.

At this time, there is no general rule for modeling the goals' system of a DRL problem, and still it is necessary spending time in designing it for each individual problem. Since individual goals imply individual rewards, it is a decision which depends on how specific the sub-task performed by each individual agent is, and to what extent the designer is familiar with the problem and each decentralized sub-problem. If there is only a common goal, this is directly related with the global task or desired behavior and guided by the common reward function. Otherwise, if individual goals are considered, their combination must guarantee to achieve the common goal.

For instance, the common goal for the 3DMC problem is reaching the goal state at the north-east corner in the Fig. 2. Individual goals can be identified, $Agent_x$ should reach the east top, and $Agent_y$ should reach the north top.

3.3. Defining the reward functions

If no individual goals have been assigned in the former stage, this step just consists of defining a global reward function according to the common goal and the desired behavior which the DRL system is designed for. If this is not the case, individual reward functions must be designed according to each individual goal.

Design considerations for defining the global or each individual reward function are the same for classical RL systems [43]. This is the most important design step requiring experience, knowledge, and creativity. A well-known rule is that the RL agent must be able to observe or control every variable involved in the reward function $R(S, A)$. Then, the next stage of this methodology consists of determining the state spaces.

In the centralized modeling for the 3DMC problem, a global reward function is proposed as: $r = 0$ if the *common goal is reached*, $r = -1$ *otherwise*. In the DRL scheme, individual reward functions can be defined as: $r^x = 0$ if *East top is reached*, $r^x = -1$ *otherwise*, for the $Agent_x$, and $r^y = 0$ if *north top is reached*; $r^y = -1$ *otherwise*, for the $Agent_y$. In this way, each single sub-task is more specific.

3.4. Determining if the problem is fully decentralizable

The next stage in this methodology consists of determining if it is necessary and/or possible to decentralize the state space too. In Section 3.1 it was determined that at least the action space will be split according to its number of dimensions. Now we are going to determine if it is also possible to simplify the state space using one separate state vector per each individual agent. This is the particular situation in which a DRL architecture offers the maximum benefit.

Proposition 2. *A DRL problem is fully decentralizable if not all the state information is relevant to all the agents, thus, individual state vectors can be defined for each agent.*

If a system is not fully decentralizable, and it is necessary that all the agents observe the whole state information, the same state vector must be used for all the individual agents, and will be called a *joint state vector*. However, if a system is fully decentralizable, the next stage is to determine which state variables are relevant to each individual agent. This decision depends on the transition function T^m of each individual goal defined as in Section 3.2, as well as on each individual reward function designed as in Section 3.3. For example, for a classical RL system, the definition of the state space must include every state variable involved in the reward function, as well as other states relevant to accomplishing the assigned goal.

Note that individual reward functions do not imply individual state spaces per agent. For instance, the 3DMC example can be designed with those two individual rewards (r^x and r^y) defined as in Section 3.3, observing the full joint state space $[x, \dot{x}, y, \dot{y}]$. Also, note that state space could be reduced for practical effects, $Agent_x$ could eventually work without observing $\dot{y}$ speed, as well as $Agent_y$ without observing $\dot{x}$ speed. So, a simplified 3DMC could be modeled as a fully decentralized problem with two individual agents with their own independent state vectors, $s^x = [x, \dot{x}, y]$, $s^y = [x, y, \dot{y}]$. Furthermore, we have implemented an extreme case with incomplete observations in which $s^x = [x, \dot{x}]$, $s^y = [y, \dot{y}]$. Implementation details as well as experimental results can be seen in Section 5.1.

3.5. Completing RL single modelings

Once the global DRL modeling has been defined and the tuples state, action, and reward $[S^m, A^m, R^m]$ are well identified per every agent $m = 1, \cdots, M$, it is necessary to complete each single RL modeling. Implementation and environmental details such as ranges and boundaries of features, terminal states, and reset conditions must be defined, as well as RL algorithms and parameters selected. If individual sub-tasks and their goals are well identified, modeling each individual agent implies the same procedure as in a classical RL system. Some problems can share some of these design details among all or some of their DRL agents. This is one of the most interesting aspects of using a DRL system: flexibility to implement completely different modelings, RL algorithms, and parameters per each individual agent; or the simplicity of just using the same scheme for all the agents.

An important design issue at this stage, is choosing the RL algorithm to be implemented per each agent properly. Considerations for modeling a classical RL single agent are also applicable here. For instance, for a discrete state-action space problem it could be more useful to use algorithms like tabular Q-Learning [53] or R-MAX [4]; for a continuous state and discrete action space problem, a SARSA with function approximation [43] might be more useful; for a continuous state-action space problem, a Fuzzy Q-Learning [13] or an actor critic scheme [15] could be more convenient. These cases are only examples to give an idea about the close relationship between modeling and designing classical RL agents versus each individual DRL agent. As already mentioned, differences are based on determining terminal states separately, resetting conditions, and establishing environment limitations, among other design settings, which can be different among agents and must be well set to coordinate the parallel learning procedure under the joint environmental conditions. Of course, depending on the particular problem, the designer has to model and define the most convenient scheme. Also note that if well-known RL algorithms are used, no extra considerations must be taken into account in designing and modeling a DRL system. Thereby, a strong background in MAS and/or MAL is not necessary.

3.6. Summary

A methodology for modeling and implementing a DRL system has been presented in this section by following a five stage design procedure. It is important to mention that some of these stages must not necessarily be applied in the same order in which they were presented. That depends on the particular problem and its properties. For instance, for some problems it could be necessary or more expeditious to define the state spaces in advance in Stage 3.4 rather than to determine individual goals in Stage 3.2. However, this is a methodology which guides the design of DRL systems in a general way. A block diagram of the proposed procedure is shown in Fig. 3.

4. Multi-agent learning methods

In this section, we examine some practical DRL algorithms to show that the benefits of MAS are also applicable to complex and real-world problems (such as robotic platforms) by using DRL systems. For this, we have implemented and tested some relevant MAL methods from state-of-the-art which accomplish the three basic requirements of our interest: (i) no

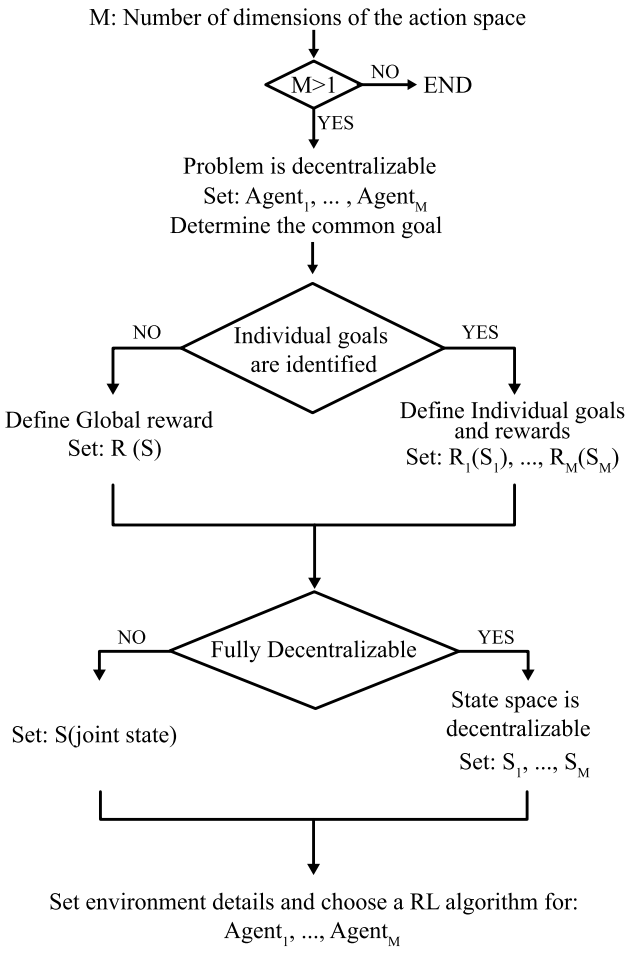

Fig. 3. Proposed procedure for modeling a DRL problem.

prior coordination, (ii) no teammates models estimation, and (iii) non-exponential increasing of computational resources when more agents are added. Based on our initial experiments, a brief note on our preliminary results from the selected methods is provided below:

(a) *Distributed Q-Learning* [21]: asymptotic convergence was not observed for all the trials, which can be explained by the stochasticity of the studied scenarios.
(b) *Frequency Adjusted Multi-Agent Q-Learning* [19]: we observed poor performance since parameter β is too sensitive and thus it was difficult to adjust; however, the idea of an adjustable learning rate from the Boltzmann probability distribution is of relevant interest.
(c) Adaptations of the *Infinitesimal Gradient Ascent* algorithm (IGA) [41] and the *Win or Learn Fast* (WoLF) principle [3]: not a trivial implementation in the case of more than two agents and non-competitives environments; however, a cooperative and variable learning rate is a promising approach.
(d) *Lenient Frequency Adjusted Q-learning* (LFAQ) [1]: it exposed poor performance due to both the tabular nature to handle lenience, and the high complexity to adjust individual FA parameters.
(e) *Independent Multi-Agent Q-Learning* without sharing information (e.g., the one reported by Sen, Sekaran, and Hale [40]): it mostly showed asymptotic convergence but poor final performances.
(f) *Lenient Multi-Agent Reinforcement Learning* [35]: it showed asymptotic convergence when applied to multi-state DRL problems.

From the above, in the present study we have decided to use the following three algorithms: (i) ***Independent DRL*** (DRL-Independent), similar to (a) but implemented with SARSA; (ii) ***Lenient Multi-Agent Reinforcement Learning*** (DRL-Lenient), as in (f) but extended to multi-state DRL problems; and (iii) ***Cooperative Adaptive Learning Rate*** (DRL-CA) algorithm, our proposed approach, inspired by (b) and (c). These approaches will be addressed in detail in the following subsections, and the corresponding performance will be discussed in Section 5.

4.1. DRL-Independent

This scheme aims for applying single-agent RL methods to the MARL task, and it does not consider any of the following features: cooperation or coordination among agents; adaptation to the other agents; estimated models of their policies; special action-selection mechanisms, such as communication among agents, prior knowledge, etc. The computational complexity of this DRL scheme is the same as that for a single RL agent (e.g., a Q-Learner).

According to the MAL literature, a single-agent RL can be applicable to stochastic games, although success is not necessarily guaranteed as the non-stationarity of the MARL problem invalidates most of the single-agent RL theoretical guarantees. However, it is considered a practical method due to its simplicity, and it has been used in several applications to robot systems [7,33,28]. The implementation of this scheme is presented in Algorithm 1, which depicts an episodic MA-SARSA algorithm for continuous states with Radial Basis Function (RBF) approximation [37], and ϵ-greedy exploration [43], where a learning system is modeled with an $M-dimensional$ action space and M single SARSA learners acting in parallel.

Algorithm 1 is described for the most general case of a fully-decentralized system with individual rewards, where states and rewards are annotated as s^m and r^m respectively, but it is also possible to implement a joint state vector or common reward DRL systems. In addition, note that RL parameters could have been defined separately per agent (α^m, γ^m), which is one of the DRL properties remarked in Section 2.3, but in Algorithm 1 they appear unified just for the sake of simplicity.

Algorithm 1 DRL-Independent: MA-SARSA with RBF approximation and ϵ-greedy exploration.

Parameters:
M ▷ Number of decentralized agents
α ▷ Learning rate $\in (0, 1]$
γ ▷ Discount factor $\in (0, 1]$
Φ^m ▷ Size of the feature vector ϕ^m of $agent_m$, where $m = 1, \cdots, M$
Inputs:
$S^1, \cdots, S^M$ ▷ State space of each agent
$A^1, \cdots, A^M$ ▷ Action space of each agent
Initialize $\vec{\theta}^m$ arbitrarily for each agent $m = 1, \cdots, M$
procedure FOR EACH EPISODE:
  for all agent $m \in M$ **do**
    $a^m, s^m \leftarrow$ Initialize state and action
  end for
  repeat for each step of episode:
    for all agent $m \in M$ **do**
      Take action $a = a^m$ from current state $s = s^m$
      Observe reward r^m, and next state $s' = s'^m$
      $urnd \leftarrow$ a uniform random variable $\in [0, 1]$
      if urnd $> \epsilon$ **then**
        for all action $i \in A^m(s')$ **do**
          $Q_i \leftarrow \sum_{j=1}^{\Phi^m} \theta_i^m(j)\phi_{s'}^m(j)$
        end for
        $a' \leftarrow \text{argmax}_i Q_i$
      else
        $a' \leftarrow$ a random action $\in A^m(s')$
      end if
      $Q as = \sum_{j=1}^{\Phi^m} \theta_a^m(j)\phi_s^m(j)$
      $Q as' = \sum_{j=1}^{\Phi^m} \theta_{a'}^m(j)\phi_{s'}^m(j)$
      $\delta \leftarrow r^m + \gamma\, Q as' - Q as$
      $\theta_a^m \leftarrow \theta_a^m + \alpha\delta\phi_s^m$
      $s^m \leftarrow s', a^m \leftarrow a'$
    end for
  until Terminal condition
end procedure

4.2. DRL-Lenient

Originally proposed by Panait et al. [35], the argument of lenient learning is that each agent should be lenient with its teammates at early stages of the concurrent learning processes. Later, Panait, Tuyls, and Luke [36] suggested that the agents should ignore lower rewards (observed upon performing their actions), and only update the utilities of actions based on the higher rewards. This can be achieved in a simple manner if the learners compare the observed reward with the estimated utility of an action and update the utility only if it is lower than the reward, namely, by making use of the rule

$$\textbf{if } (U_{a^*} \leq r) \textbf{ or } urnd < 10^{-2} + \kappa^{-\beta\tau_{a^*}} \textbf{ then } U_{a^*} \leftarrow \alpha U_{a^*} + (1-\alpha)r, \tag{3}$$

where $urnd \in [0, 1]$ is a random variable, κ is the lenience exponent coefficient, and $\tau(a^*)$ is the lenience temperature of the selected action. Lenience may be reduced as learning progresses and agents start focusing on a solution that becomes

more critical with respect to joint rewards (ignoring fewer of them) during advanced stages of the learning process, which can be incorporated in Eq. (3) by using a discount factor β each time that action is performed.

Lenient learning was initially proposed in state-less MA problems. According to Troost et al. [48] and Schuitema [39], a multi-state implementation of Lenient Q-learning can be accomplished by combining the Q-Learning update rule (i.e. Eq. (1)) with the optimistic assumption proposed by Lauer and Riedmiller [21]. Accordingly, the action-value function is updated optimistically at the beginning of the learning trial, taking into account the maximum utility previously received along with each state-action pair visited. Then, lenience toward other agents is refined smoothly, returning to the original update function (this is, Eq. (1)):

$$Q(s,a) \leftarrow \begin{cases} Q(s,a) + \alpha\delta, & \text{if } \delta > 0 \text{ or } urnd > \ell(s,a), \\ Q(s,a), & \text{otherwise,} \end{cases} \tag{4}$$

with the state-action pair dependent lenience $\ell(s,a)$ defined as

$$\ell(s,a) = 1 - \exp(-\kappa\tau(s,a)),$$
$$\tau(s,a) \leftarrow \beta\tau(s,a),$$

where κ is the lenience coefficient, and $\tau(s,a)$ is the lenience temperature of the state action pair (s,a), which decreases with a discount factor β each time the state-action pair is visited.

In our study, we implement lenient learning by adapting the update rule (4) to multi-state, stochastic, continuous state-action DRL problems, as reported by Troost et al. [48] and Schuitema [39]. The DRL-Lenient algorithm presented in Algorithm 2, which is implemented by replacing traces, incorporates a tabular MA-SARSA(λ) method, and uses softmax action selection from Sutton and Barto [43].

In Algorithm 2, individual temperatures are managed separately by each state-action pair. These temperatures (line 20) are used to later compute the Boltzmann probability distribution Pa (line 26), which is the basis for the softmax action selection mechanism. Note that only the corresponding temperature $\tau(s_t, a_i)$ is decayed in line 29 after the state-action pair (s_t, a_i) is visited. This is a difference with respect to the usual softmax exploration which uses a single temperature for the entire learning process. Value function is updated only if the learning procedure is either optimistic or lenient, otherwise it is not updated. It is either optimistically updated whenever the last performed action increases the current utility function, or leniently updated if the agent has explored that action sufficiently. Since lenience (line 30) is also computed from temperature, every state-action pair has an individual lenience degree as well. The agent is more lenient (and it thus ignores low rewards) if the temperature associated with the current state-action pair is high. Such a leniency is reduced as long as its respective state-action pair is visited; in that case, the agent will tend to be progressively more critical in refining the policy.

In order to extend DRL-Lenient to continuous states, it is necessary to implement a function approximation strategy for the lenient temperature $\tau(s,a)$, the eligibility traces $e(s,a)$, and the action-value functions. Following a linear gradient-descent strategy with RBF-features, similar to that presented in Algorithm 1, function approximations can be expressed as:

$$e_a \leftarrow e_a + \phi_s, \tag{5a}$$
$$\tau(s,a) = \sum_{j=1}^{\Phi} \tau_a(j)\phi_s(j), \tag{5b}$$
$$\tau_a \leftarrow \tau_a - (1-\beta)\tau(s,a)\phi_s, \tag{5c}$$
$$\delta \leftarrow r + \gamma \sum_{j=1}^{\Phi} \theta_{a'}(j)\phi_{s'}(j) - \sum_{j=1}^{\Phi} \theta_a(j)\phi_s(j), \tag{5d}$$
$$\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}, \tag{5e}$$
$$\vec{e} \leftarrow \gamma\lambda\vec{e}, \tag{5f}$$

where Φ is the size of the feature vector ϕ_s. Equations (5a), (5c), (5d), (5e) and (5f) would approximate lines 19, 29, 28, 33 and 36, respectively. For practical implementations, τ_a must be set between $(0, 1)$.

4.3. DRL-CA

In this paper, we introduce the DRL Cooperative Adaptive Learning Rate algorithm (DRL-CA), which mainly takes inspiration from the MARL approaches with a variable learning rate [3], and Frequency Adjusted Q-Learning (FAQL) [19]. We have used the idea of a variable learning rate from the WoLF principle [3] and the IGA algorithm [41], in which agents learn quickly when losing, and cautiously when winning. The WoLF-IGA algorithm requires knowing the actual distribution of the

Algorithm 2 DRL-Lenient: SARSA(λ) with softmax action selection.

Parameters:
M ▷ Number of decentralized agents
N^m ▷ Number of actions of $agent_m$, where $m = 1, \cdots, M$
λ ▷ Eligibility trace decay factor $\in [0, 1)$
κ ▷ Lenience coefficient
β ▷ Lenience discount factor $\in [0, 1)$
Inputs:
$S^1, \cdots, S^M$ ▷ State space of each agent
$A^1, \cdots, A^M$ ▷ Action space of each agent
for all agent $m \in M$ **do**
  for all (s^m, a^m) **do**
    Initialize:
    $Q^m(s^m, a^m) = 0$, $e^m(s^m, a^m) = 0$, and $\tau^m(s^m, a^m) = 1$
  end for
  Initialize state and action s^m, a^m
end for
repeat
  for all agent $m \in M$ **do**
    Take action $a = a^m$ from current state $s = s^m$
    Observe reward r^m, and next state $s' = s'^m$
    $e^m(s, a) \leftarrow 1$
    $min\tau \leftarrow \kappa(1 - \min_{action}^{N^m}{}_{i=1}(\tau^m(s, a_i)))$
    $maxQ\,v \leftarrow \max_{action}^{N^m}{}_{i=1}(Q^m(s, a_i))$
    for all action $i \in A^m(s')$ **do**
      $Vqa_i \leftarrow \exp(min\tau(Q^m(s, a_i) - maxQ\,v))$
    end for
    $Pa = [Pa_1, \cdots, Pa_{N^m}]$ ▷ Define probability distribution per-action at state s
    $Pa \leftarrow \frac{Vqa}{\sum_{i=1}^{N^m} Vqa_i}$
    Choose action $a' = a_{i^*} \in \{1, \cdots, N^m\}$ ▷ at random using probability distribution $[Pa_1, \cdots, Pa_{N^m}]$
    $\delta \leftarrow r^m + \gamma Q^m(s', a') - Q^m(s, a)$
    $\tau^m(s, a) \leftarrow \beta\tau^m(s, a)$
    $\ell(s, a) = 1 - \exp(-\kappa\tau^m(s, a))$
    if $\delta > 0$ — $urnd > \ell(s, a)$ **then**
      for all (s, a) **do**
        $Q^m(s, a) \leftarrow Q^m(s, a) + \alpha\delta e^m(s, a)$
      end for
    end if
    $e^m \leftarrow \gamma\lambda e^m$
    $s^m \leftarrow s'; a^m \leftarrow a'$
  end for
until Terminal condition

actions the other agents are playing, in order to determine if an agent is winning. This requirement is hard to accomplish for some MA applications in which real-time communication is a limitation (e.g., decentralized multi-robot systems), but it is not a major problem for DRL systems performing single robot behaviors. Thus, DRL-CA uses a cooperative approach to adapt the learning rate, sharing the actual distribution of actions per-each agent. Unlike the original WoLF-IGA, where gradient ascent is derived from the expected pay-off, or unlike the current utility function from the update rule [3], DRL-CA directly uses the probability of the selected actions, having a common normalized measure of partial quality of the policy performed per agent. This idea is similar to FAQ-Learning [19], in which the Q update rule

$$Q_i(k+1) \leftarrow Q_i(k) + \min\left(\frac{\beta}{Pa_i}, 1\right)\alpha[r + \gamma \max_j Q_j(k) - Q_i(k)] \tag{6}$$

is modified by the adjusted frequency parameter $(\min(\beta/x_i, 1))$. In our DRL-CA approach, we replace such term by a cooperative adaptive factor ς defined as

$$\varsigma = 1 - \min_{agent}^{M}{}_{m=1} Pa^{*,m} \tag{7}$$

The main principle of DRL-CA is supported on this cooperative factor that adapts a global learning rate on-line, which is based on a simple estimation of the partial quality of the joint policy performed. So, ς is computed from the probability of selected action (Pa^*), according to the "weakest" among the M agents.

A variable learning rate based on the gradient ascent approach presents the same properties as an algorithm with an appropriately decreasing step size [41]. In this way, DRL-CA shows a decreasing step size if a cooperative adaptive factor ς such as (7) is used. We refer to this decremental variation as ***DRL-CAdec***. So, an agent should adapt quickly during the

early learning process, trying to collect experience and learn fast while there is a mis-coordinated joint policy. In this case, we have that $\varsigma \to 1$ and the learning rate tends to α. Once the agents progressively obtain better rewards, they should be cautious since the other players are refining their policies and, eventually, they will explore unknown actions which can produce temporal mis-coordination. In this case, we have $\varsigma \to 0$ and a decreasing learning rate, while better decisions are being made. Note that DRL-CAdec acts contrarily to the DRL-Lenient principle.

We also introduce the ***DRL-CAinc***, a variation in which a cooperative adaptive factor increases during the learning process if a coordinated policy is learned gradually. This variation uses

$$\varsigma = \min_{agent} \min_{m=1}^{M} Pa^{*,m} \tag{8}$$

instead of (7). Here, a similar lenient effect occurs, and the agents update their utilities cautiously during the early learning process, being lenient with weaker agents while they learn better policies. In this case, ς starts from the lowest probability among all the agents, making the learning rate tend to a small but non-zero value. Once the agents are progressively obtaining better rewards, they learn and update from their coordinated joint policy. Then, in this case, $\varsigma \to 1$ and the learning rate tends toward a high value.

DRL-CAdec and DRL-CAinc show opposite principles. A detailed analysis of their properties is presented in Section 5. The common principle behind both variants is the cooperative adaptation based on the current weakest learner's performance. We also have empirically tested other cooperative adaptive factors, but they resulted in no success: (i) based on individual factors, $\varsigma^m = Pa^{*,m}$ for each $agent_m$; (ii) based on the best agent, $\varsigma = \max_m Pa^{*,m}$; and (iii) based on the mean of their qualities, $\varsigma = mean_m Pa^{*,m}$.

The chosen approach (based on the weakest agent) coordinates the learning evolution awaiting similar skills among the agents. This is possible since ς comes from a Boltzmann distribution, which is a probability always bounded between $[0, 1]$, and thus it is possible to consider ς as a measure of the current learned skill by an agent. This is desirable for the cooperation among the agents, and is an advantage over methods based on the Temporal Difference (TD) or instant reward, in which their gradients are not normalized and extra parameters must be adjusted. Concerning DRL-CAinc, the most skilled agents wait for the less skilled one, showing leniency by adapting the learning rate according to the current utility function of the weakest learner. This makes sense because the policy of the most skilled agents could change when the less skilled one improves its policy, so the agents should be cautious. Once all the agents have similar skills, the learning rate is gradually increased for faster learning while the joint policy is improved. In the case of DRL-CAdec, the less-skilled agents motivate their teammates to extract more information from the joint-environment and joint-actions, in order to find a better common decision which can quickly improve such a weak policy.

Algorithm 3 presents the DRL-CA implementation for multi-state, stochastic, continuous state-action DRL problems. It is an episodic MA-SARSA(λ) algorithm with RBF approximation and softmax action selection. The incremental cooperative adaptive factor (Eq. (8)) is calculated in line 32, and the decremental cooperative adaptive factor (Eq. (7)) is calculated in line 34.

Note that, for practical implementations in which agents have different numbers of discrete actions, each $Pa^{*,m}$ must be biased to $Pa^{*',m}$ in order to have equal initial probabilities among the individual agents, i.e. $Pa^{*',1}_{s=0} = \cdots = Pa^{*',M}_{s=0}$, and then $Pa^{*',m} = \mathcal{F}_{bias}(Pa^{*,m})$, where $\forall\ Pa^{*',m} \in [0, 1]$. A simple alternative to calculate this is by computing $Pa^{*',m} = \max(1/N^m, Pa^{*,m})$, or

$$Pa^{*',m} = Pa^{*,m} - \left[\frac{(N^m Pa^{*,m} - 1)}{(N^m(1 - N^m))} + \frac{1}{N^m} \right] \tag{9}$$

which is a more accurate approach. This bias must be computed after line 28, and then σ in line 32 must be computed by using $Pa^{*',m}$ instead of the non-biased $Pa^{*,m}$.

Note that both Algorithms 2 and 3 have been described with a softmax action selection mechanism. Other exploration methods such as ϵ-greedy can be easily implemented, but it must be taken into account that both methods DRL-Lenient and DRL-CA are based on the Boltzmann probability distribution, Pa, which must bee calculated as well. However, this only requires on-line and temporary computations, and no extra memory consumption.

5. Experimental validation

In order to validate MAS benefits and properties of the DRL systems, four different problems have been carefully selected: the 3DMC, a three-Dimensional extension of the mountain car problem [46]; the SCARA-RTG, a SCARA robot generating a real-time trajectory for navigating towards a 3D goal position [31]; the *Ball-Pushing* performed with a differential drive robot [28]; and the soccer *Ball-Dribbling* task [25]. The 3DMC and SCARA-RTG are well known and are already proposed test-beds. The Ball-Dribbling and Ball-Pushing problems are noisy and stochastic real-world applications that have been tested already with physical robots.

The problem descriptions and results are presented in a manner of increasing complexity. 3DMC is a canonical RL test-bed; it allows splitting the action space, as well as the state space for evaluating from a centralized system, up to a fully decentralized system with limited observability of the state space. The Ball-Pushing problem also allows carrying out a

Algorithm 3 DRL-CA: MA-SARSA(λ) with RBF approximation and Softmax action selection.

Parameters:
M ▷ Decentralized agents
N^m ▷ Number of actions of $agent_m$, where $m = 1, \cdots, M$
τ_0 ▷ Temperature
dec ▷ Temperature decay factor
Φ^m ▷ Size of the feature vector ϕ^m of $agent_m$, where $m = 1, \cdots, M$
Inputs:
$S^1, \cdots, S^M$ ▷ State space of each agent
$A^1, \cdots, A^M$ ▷ Action space of each agent
for each agent $m = 1, \cdots, M$ **do**
  Initialize: $\vec{\theta}^m = 0, \vec{e}^m = 0, \tau = \tau_0$, and $\varsigma = 1$
end for
for $episode = 1, \cdots, maxEpisodes$ **do**
  Initialize state and action s^m, a^m **for all** agent $m \in M$
  repeat for each step of episode:
    for all agent $m \in M$ **do**
      Take action $a = a^m$ from current state $s = s^m$
      Observe reward r^m, and next state $s' = s'^m$
      $e_a \leftarrow e_a + \phi_s$
      $\delta \leftarrow r^m - \sum_{j=1}^{\Phi^m} \theta_a^m(j)\phi_s^m(j)$
      $Q_i \leftarrow \sum_{j=1}^{\Phi^m} \theta_i^m(j)\phi_{s'}^m(j)$ **for all** action $i \in A^m(s')$
      $maxQ\,v \leftarrow \max_{action\ i=1}^{N^m} Q_i$
      $Vqa_i \leftarrow \exp\left(\frac{(Q_i - maxQ\,v)}{(1+\tau)}\right)$ **for all** action $i \in N^m$
      $Pa = [Pa_1, \cdots, Pa_{N^m}]$ ▷ probability distribution per-action at state s
      $Pa \leftarrow Vqa / \sum_{i=1}^{N^m} Vqa_i$
      Choose action $a' = a_{i^*} \in \{1, \cdots, N^m\}$ ▷ at random using probability distribution $[Pa_1, \cdots, Pa_{N^m}]$
      $\delta \leftarrow \delta + \gamma\, Q_{i^*}$
      $\vec{\theta}^m \leftarrow \vec{\theta}^m + \varsigma\alpha\delta\vec{e}^m$
      $\vec{e} \leftarrow \gamma\lambda\vec{e}$
      $Pa^{*,m} \leftarrow Pa_{a'}$ ▷ Boltzmann probability of the selected action
      $s^m \leftarrow s'; a^m \leftarrow a'$
    end for
    $\tau = \tau_0 \exp(\text{-}decepisode/maxEpisodes)$
    $\varsigma = \min_{agent\ m=1}^{M} (Pa^{*,m})$ ▷ CAinc variation
    if CAdec variation **then**
      $\varsigma = 1 - \varsigma$
    end if
  until Terminal condition
end for

performance comparison between a centralized and a decentralized scheme. The best CRL and DRL learned policies are transferred and tested with a physical robot. The Ball-Dribbling and SCARA-RTG problems are more complex systems (implemented with 3 and 4 individual agents respectively). Ball-dribbling is a very complex behavior which involves three parallel sub-tasks in a highly dynamic and non-linear environment. The SCARA-RTG has four joints acting simultaneously in a 3-Dimensional space, in which the observed state for the whole system is only the error between the current end-effector position, $[x, y, z]$, and a random target position.

Some relevant parameters of the RL algorithms implemented are optimized by using a customized version of the hill-climbing method. It is carried out independently for each approach and problem tested. Details about the optimization procedure and the pseudo-code of the implemented algorithm can be found in Appendix A. Finally, 25 runs are performed by using the best parameter settings obtained in the optimization procedure. Learning evolution results are plotted by averaging those 25 runs, and error bars show the standard error. In addition, the averaged final performance is also measured: it considers the last 10% of the total learning episodes.

A description of each problem tested and some implementation and modeling details are presented in the next subsections, following the methodology described in Section 3. The experimental results and analysis are then discussed. All the acronyms of the implemented methods and problems are listed in Table 1. We used the following terminology: CRL means a Centralized RL scheme; DRL-Ind is an independent learners scheme implemented without any kind of MA coordination; DRL-CAdec, DRL-CAinc, and DRL-Lenient are respectively a DRL system coordinated with Decremental Cooperative Adaptation, Incremental Cooperative Adaptation, and a Lenient approach. In the case of the 3DMC, CRL-5a and CRL-9a are Centralized RL schemes implemented with 5 actions (the original 3DMC modeling [46]) and 9 actions (our extended version) respectively. ObsF and ObsL are Full Observability and Limited observability of the joint state space respectively. In the case of the Ball-Pushing problem, DRL-Hybrid is a hybrid DRL-Ind scheme implemented with a SARSA(λ) + a Fuzzy Q-Learning RL algorithm without any kind of MAS coordination (please see a detailed description in subsection 5.2). In the case of the Ball-Dribbling problem, DRL-Transfer is a DRL system accelerated by using the NASh transfer knowledge learn-

Table 1
Experiment's acronyms and their optimized parameters.

Acronym	Optimized parameters
3DMC	
CRL-5a	$\alpha = 0.25, \lambda = 0.95, \epsilon = 0.06$
CRL-9a	$\alpha = 0.20, \lambda = 0.95, \epsilon = 0.06$
DRL-ObsF-Ind	$\alpha = 0.25, \lambda = 0.80, \epsilon = 0.06$
DRL-ObsF-CAdec	$\alpha = 0.15, \lambda = 0.90, \epsilon = 0.05$
DRL-ObsF-CAinc	$\alpha = 0.20, \lambda = 0.80, \epsilon = 0.06$
DRL-ObsF-Lenient	$\alpha = 0.10, \lambda = 0.95, \epsilon = 0.04, \kappa = 3.5, \beta = 0.8$
DRL-ObsL-Ind	$\alpha = 0.20, \lambda = 0.95, \epsilon = 0.06$
DRL-ObsL-CAdec	$\alpha = 0.15, \lambda = 0.95, \epsilon = 0.05$
DRL-ObsL-CAinc	$\alpha = 0.30, \lambda = 0.95, \epsilon = 0.02$
DRL-ObsL-Lenient	$\alpha = 0.15, \lambda = 0.95, \epsilon = 0.10, \kappa = 3, \beta = 0.75$
Ball-Pushing	
CRL	$\alpha = 0.50, \lambda = 0.90, \tau_0 = 2, dec = 7$
DRL-Ind	$\alpha = 0.30, \lambda = 0.90, \tau_0 = 1, dec = 10$
DRL-CAdec	$\alpha = 0.40, \lambda = 0.95, \tau_0 = 1, dec = 10$
DRL-CAinc	$\alpha = 0.30, \lambda = 0.95, \tau_0 = 5, dec = 13$
DRL-Lenient	$\alpha = 0.30, \lambda = 0.95, \ \kappa = 1, \ \beta = 0.7$
DRL-Hybrid	$\alpha = 0.30, \lambda = 0.95$, greedy
Ball-Dribbling	
CRL	$\alpha = 0.50, \lambda = 0.90, \epsilon = 0.3, dec = 10$
DRL-Ind	$\alpha = 0.50, \lambda = 0.90, \tau_0 = 70, dec = 6$
DRL-CAdec	$\alpha = 0.10, \lambda = 0.90, \tau_0 = 20, dec = 8$
DRL-CAinc	$\alpha = 0.30, \lambda = 0.90, \tau_0 = 70, dec = 11$
DRL-Lenient	$\alpha = 0.10, \lambda = 0.90, \ \kappa = 1.5, \beta = 0.9$
DRL+Transfer	Final performance taken from Leottau et al. [26]
RL-FLC	Final performance taken from Leottau et al. [25]
eRL-FLC	Final performance taken from Leottau et al. [27]
SCARA-RTG	
DRL-Ind	$\alpha = 0.3, \epsilon = 0.01$
DRL-CAdec	$\alpha = 0.3, \epsilon = 0.01$
DRL-CAinc	$\alpha = 0.3, \epsilon = 0.01$
DRL-Lenient	$\alpha = 0.3, \epsilon = 0.01, \kappa = 2.0, \beta = 0.8$

ing approach [26]; RL-FLC is an implementation reported by Leottau, Celemin, and Ruiz del Solar [25], which combines a Fuzzy Logic Controller (FLC) and an RL single agent; and eRL-FLC is an enhanced version of RL-FLC (please see their detailed descriptions in Subsection 5.3).

5.1. Three-dimensional mountain car

Mountain car is one of the canonical RL tasks in which an agent must drive an under-powered car up a mountain to reach a goal state. In the 3D modification originally proposed by Taylor and Stone [46], the mountain's curve is extended to a 3D surface as is shown in Fig. 2.

5.1.1. Centralized modelings

CRL-5a: The state has four continuous state variables: $[x, \dot{x}, y, \dot{y}]$. The positions (x, y) have the range of $[-1.2, 0.6]$ and the speeds $(\dot{x}, \dot{y})$ are constrained to $[-0.07, 0.07]$. The agent selects from five actions: {Neutral, West, East, South, North}. West and East on $\dot{x}$ are modified by -0.001 and $+0.001$ respectively, while South and North on $\dot{y}$ are modified by -0.001 and $+0.001$ respectively. On each time step $\dot{x}$ is updated by $-0.025(\cos(3x))$ and $\dot{y}$ is updated by $-0.025(\cos(3y))$ due to gravity. The goal state is $x \geq 0.5$ and $y \geq 0.5$. The agent begins at rest at the bottom of the hill. The reward is -1 for each time step until the goal is reached, at which point the episode ends and the reward is 0. The episode also ends, and the agent is reset to the start state, if the agent fails to find the goal within 5000 time steps.

CRL-9a: The original centralized modeling (CRL-5a) [46] limits the agent's vehicle moves. It does not allow acting onto both action dimensions at the same time step. In order to make this problem fully decentralizable, more realistic, and challenging, we have extended the problem, augmenting the action space to nine actions (CRL-9a), adding *{NorthWest, NorthEast, SouthWest, SouthEast}* to the original CRL-5a. Since the car is now able to move on x and y axes at the same time, $\dot{x}$, and $\dot{y}$ updates must be multiplied by $1/\sqrt{2}$ for the new four actions because of the diagonal moves.

5.1.2. Proposed decentralized modelings

We are going to follow the methodology proposed in Section 3, resuming and extending the 3DMC DRL modeling:

Stage 3.1 — Determining if the problem is decentralizable: since CRL-9a modeling is decentralizable because of its bi-dimensional action space $(\dot{x}, \dot{y})$, a decentralized approach can be adopted by selecting two independent agents: $Agent_x$ which action space is $\{Neutral, West, East\}$, and $Agent_y$ which action space is $\{Neutral, South, North\}$.

Stages 3.2 and 3.3 — Identifying individual goals and defining reward functions: individual goals are considered, reaching east top for $Agent_x$ and reaching north top for $Agent_y$. In this way, individual reward functions are defined as: $r^x = 0$ if *east top is reached*, $r^x = -1$ *otherwise*; and $r^y = 0$ if *north top is reached*, $r^y = -1$ *otherwise*.

Stage 3.4 — Determining if the problem is fully decentralizable: one of the goals of this work is evaluating and comparing the response of an RL system under different centralized–decentralized schemes. Thus, splitting the state vector is also proposed in order to have a fully decentralized system, and a very limited state observability for validating the usefulness of coordination of the presented MA DRL algorithms (Lenient and CA). In this case, $agent_x$ only state variables $[x, \dot{x}]$ can be observed, as well as $agent_y$ only $[y, \dot{y}]$. This corresponds to a very complex scenario because both agents have incomplete observations, and do not even have free or indirect coordination due to different state spaces, decentralized action spaces, and individual reward functions. Moreover, the actions of each agent directly affect the joint environment, and both of the agents' next state observations.

A description of the implemented modelings is shown below, in which X can be CAdec, CAinc, or Lenient, and RBF cores are the number of Radial Basis Function centers used per state variable to approximate action value functions as continuous functions. Please see Table 1 for the full list of acronyms.

- *CRL Original Modeling (CRL-5a):*
  Actions: $\{Neutral, West, East, South, North\}$;
  Global reward function: $r = 0$ if goal, $r = -1$ otherwise. Joint state vector: $[x, \dot{x}, y, \dot{y}]$, with $[9, 6, 9, 6]$ RBF cores per state variable respectively;
- *CRL Extended Modeling (CRL-9a):*
  Actions: $\{Neutral, West, NorthWest, North, NorthEast, East, SouthEast, South, SouthWest\}$;
  Global reward function: $r = 0$ if goal, $r = -1$ otherwise. Joint state vector: $[x, \dot{x}, y, \dot{y}]$, with $[9, 6, 9, 6]$ RBF cores;
- *DRL Full Observability (DRL-ObsF-X):*
  Actions $agent_x$: $\{Neutral, West, East\}$,
  Actions $agent_y$: $\{Neutral, South, North\}$;
  Individual reward functions: $r^x = 0$ if $x \geq 0.5$, $r^x = -1$ otherwise, and $r^y = 0$ if $y \geq 0.5$, $r^y = -1$ otherwise.
  Joint state vector: $[x, \dot{x}, y, \dot{y}]$, with $[9, 6, 9, 6]$ RBF cores;
- *DRL Limited Observability (DRL-ObsL-X):*
  Actions $agent_x$: $\{Neutral, West, East\}$,
  Actions $agent_y$: $\{Neutral, South, North\}$;
  Individual reward functions: $r^x = 0$ if $x \geq 0.5$, $r^x = -1$ otherwise, and $r^y = 0$ if $y \geq 0.5$, $r^y = -1$ otherwise.
  Individual state vectors: $agent_x = [x, \dot{x}]$, with $[9, 6]$ RBF cores; $agent_y = [y, \dot{y}]$, with $[9, 6]$ RBF cores.

Stage 3.5 — Completing RL single modelings: this is detailed in the following two subsections. Implementation and environmental details have been already mentioned in the centralized modeling description, because most of them are in common with the decentralized modeling.

5.1.3. Performance index

The evolution of the learning process is evaluated by measuring and averaging 25 runs. The performance index is the cumulative reward per episode, where $-5,000$ is the worst case and zero, though unreachable, is the best case.

5.1.4. RL algorithm and optimized parameters

SARSA(λ) with Radial Basis Function (RBF) approximation with ϵ-greedy exploration [43] was implemented for these experiments. The exploration rate ϵ is decayed by 0.99 at the end of each learning episode. The following parameters are obtained after the hill-climbing optimization procedure: learning rate (α), eligibility traces decay factor (λ), and exploration probability (ϵ). These parameters are detailed in Table 1 for each experiment. The number of Gaussian RBF cores per state variable were also optimized: 9 cores to x and y, 6 cores to $\dot{x}$ and $\dot{y}$, and a standard deviation per core of $1/2|feature_{max} - feature_{min}|/nCores$. For all the experiments $\gamma = 0.99$.

5.1.5. Results and analysis

Fig. 4 (top) shows a performance comparison between: the original implementation of 3DMC, CRL-5a; the extension of that original problem in which 9 actions are considered, CRL-9a; a decentralized scheme with full observability of the joint space state, DRL-ObsF-Ind; and a decentralized scheme with limited observability, DRL-ObsL-Ind. Please remember that the performance index starts from $-5,000$ and it improves toward zero. Table 2 shows averaged final performances. Our results for CRL-5a converge considerably faster than the results presented by Taylor and Stone [46], which could be due to parameter optimization, and because we have implemented an RBF approach instead of CMAC for continuous state generalization. CRL-9a converges more slowly than the original one, as is expected because of the augmented action space. Note that DRL-ObsF-Ind speeds-up convergence and outperforms both centralized schemes. On the other hand, DRL-ObsL-Ind achieves a

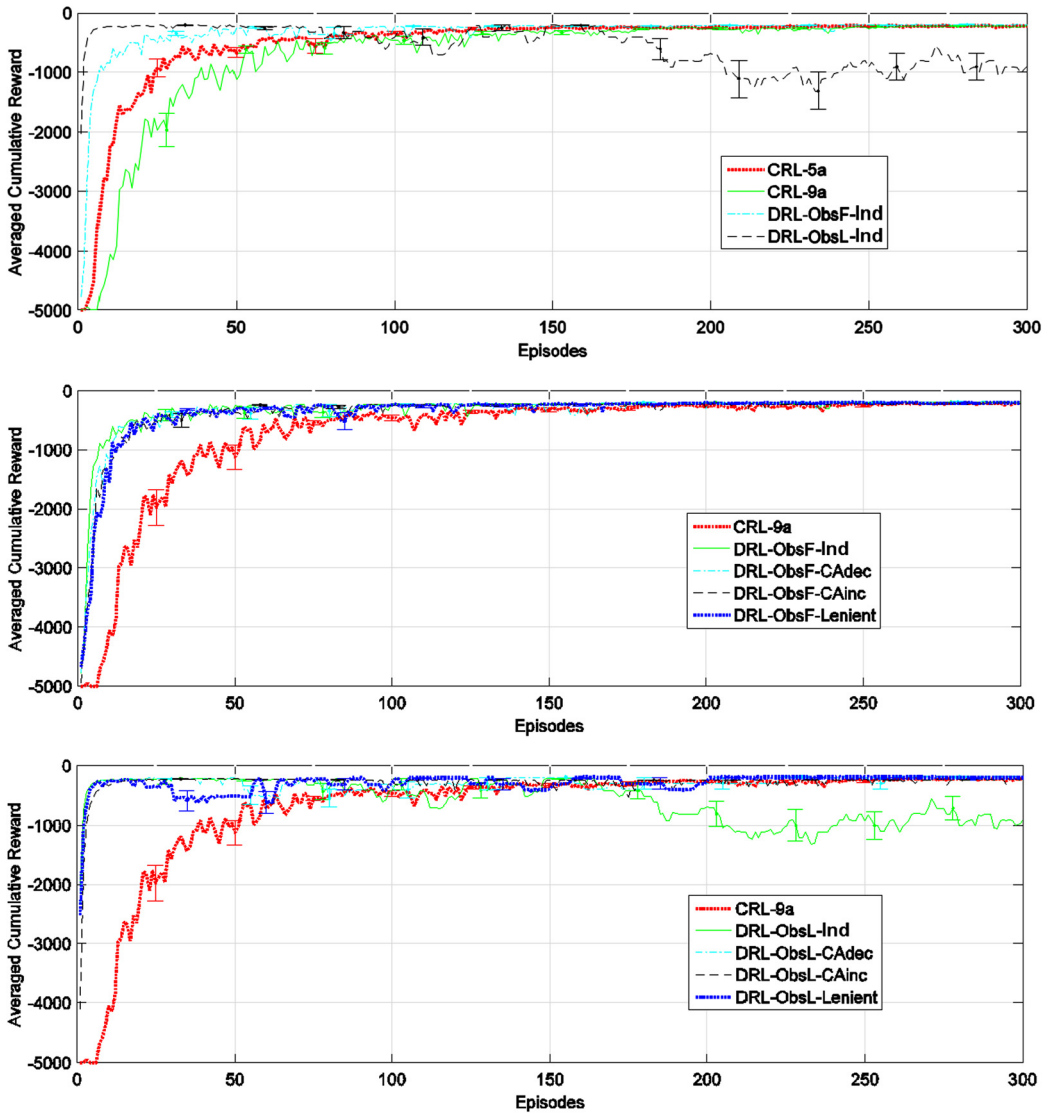

Fig. 4. 3DMC learning evolution plots: centralized vs. decentralized approaches (top); centralized vs. decentralized approaches with full observability of the joint state space (middle); centralized vs. decentralized approaches with limited observability (bottom). (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

good performance quickly but is not stable during the whole learning process due to ambiguity between observed states and lack of coordination among the agents. However, it opens a question about potential benefits of DRL implementations with limited or incomplete state spaces which is discussed below.

Regarding computational resources, from the optimized parameters definition presented above, the DRL-ObsF-Ind scheme uses two Q functions which consume $2 \cdot 9 \cdot 6 \cdot 9 \cdot 6 \cdot 3 = 17496$ memory cells, versus the $9 \cdot 6 \cdot 9 \cdot 6 \cdot 9 = 26244$ of its CRL-9a counterpart; and DRL-ObsF-Ind consumes 1/3 less memory. Moreover, we measured the elapsed time of both learning process along the 25 performed runs, and founds that the DRL took 0.62 hour, while the CRL took 0.97 hour. We also measured only the action-selection+Q-function-update elapsed times, obtaining an average of 306.81 seconds per run for the DRL, being 1.43 times faster than the CRL scheme, which took 439.59s. These times are referential; experiments with an Intel® Core™ i7-4774 CPU @ 3.40 GHz with 4 GB in RAM were performed. Note than even for this simple problem with only two agents, there are considerable memory consumption and processing time savings.

Fig. 4 (middle) shows a performance comparison between schemes implemented considering full observability (ObsF) of the joint space state, these schemes are: the same response of CRL-9a presented in Fig. 4 (top); once again the DRL-ObsF-Ind; a Decremental Cooperative Adaptive DRL-ObsF-CAdec scheme; an Incremental Cooperative Adaptive DRL-ObsF-CAinc scheme; and, a DRL-ObsF-Lenient implementation. As noticed in Fig. 4, all the DRL systems accelerate the asymptotic convergence considerably and outperform the CRL one. Note also that all the DRL systems show similar learning times, while in Table 2, DRL-ObsF-CAdec shows the best performance, overcoming the −200 performance threshold with DRL-ObsF-Lenient.

Table 2
3DMC performances (these improve toward zero).

Approach	Performance
DRL-ObsF-CAdec	−190.19
DRL-ObsF-Lenient	−196.00
DRL-ObsF-Ind	−207.35
DRL-ObsF-CAinc	−216.64
DRL-ObsL-CAinc	−186.59
DRL-ObsL-Lenient	−197.12
DRL-ObsL-CAdec	−231.30
DRL-ObsL-Ind	−856.60
CRL-9a	−219.72
CRL-5a	−217.58

Fig. 4 (bottom) shows a performance comparison between schemes implemented considering limited observability (ObsL) of the joint space state, these schemes are: CRL-9a; DRL-ObsL-Ind; a Decremental Cooperative Adaptive DRL-ObsL-CAdec scheme; an Incremental Cooperative Adaptive DRL-ObsL-CAinc scheme; and a DRL-ObsL-Lenient implementation. Benefits of proposed Lenient and CA algorithms are more noticeable in these experiments, in which the DRL-ObsL-Ind scheme without coordination did not achieve a stable final performance. With the exception of DRL implementation (green line), all the DRL systems have dramatically accelerated convergence times regarding the CRL scheme. This is empirical evidence of proposed MAS based algorithm benefits (CAdec, CAinc, and Lenient), even if incomplete observations are used. These benefits are not evident for those experiments with full observation, in which convergence time and performance are similar to the DRL-Ind scheme. DRL-ObsL-Lenient indirectly achieves a coordinated policy. Although for this particular case leniency is not directly involved in the ϵ-greedy action-selection mechanism, it is involved during the action-value function updating, which of course, affects the action-selection mechanism afterwards. On the other hand, DRL-ObsL-CAdec collects experience and, while a coordinated policy is gradually reached, the learning rate is decreased and the action-value function updates have progressively less weight. It just avoids the poor final performance of the DRL-ObsL-Ind scheme. Also DRL-ObsL-CAinc achieves a good performance; it has a similar effect to that of the Lenient algorithm. Also, note in Table 2 that DRL-ObsL-CAinc and DRL-ObsL-Lenient outperform the −200 threshold, even beating its DRL-ObsF counterparts, and beating the DRL-ObsF-Ind as well. This is an interesting result, taking into account DRL-ObsL schemes are able to reach similar performance as the DRL-ObsF-CAdec and DRL-ObsF-Lenient, the best schemes implemented with full observability.

5.2. Ball-Pushing

We consider the Ball-Pushing behavior, a basic robot soccer skill similar to that studied by Takahashi and Asada [44] and Emery and Balch [12]. A differential robot player attempts to push the ball and score a goal. The MiaRobot Pro is used for this implementation (see Fig. 5). In the case of a differential robot, the complexity of this task comes from its non-holonomic nature, limited motion and accuracy, and especially the highly dynamic and non-linear physical interaction between the ball and the robot's irregular front shape. The description of the desired behavior will use the following variables: $[v_l, v_w]$, the velocity vector composite by linear and angular speeds; a_w, the angular acceleration; γ, the robot-ball angle; ρ, the robot-ball distance; and, ϕ, the robot-ball-target complementary angle. These variables are shown in Fig. 5 at the left, where the center of the goal is located in $\oplus$, and a robot's egocentric reference system is considered with the x axis pointing forwards.

The RL procedure is carried out episodically. After a reset, the ball is placed in a fixed position 20 cm in front of the goal, and the robot is set at a random position behind the ball and the goal. The successful terminal state is reached if the ball crosses the goal line. If the robot leaves the field, it is also considered a terminal state. The RL procedure is carried out in a simulator, and the best learned policy obtained between the 25 runs for the CRL and DRL-Ind implementations is directly transferred and tested on the MiaBot Pro robot in the experimental setup.

5.2.1. Centralized modeling

For this implementation, proposed control actions are twofold $[v_l, a_w]$, the requested linear speed and the angular acceleration, where $A^{a_w} = [positive, neutral, negative]$. Our expected policy is to move fast and push the ball toward the goal; that is, to minimize ρ, γ, ϕ, and to maximize v_l. Thus, this centralized approach considers all possible action combinations $\mathcal{A} = A^{v_l} A^{a_w}$ and the robot learns $[v_l, a_w]$ actions from the observed joint state $[\rho, \gamma, \phi, v_w]$, where $[v_w = v_{w(k-1)} + a_w]$. States and actions are detailed in Table 3.

5.2.2. Decentralized modeling

Stage 3.1 — Determining if the problem is decentralizable: the differential robot velocity vector can be split into two independent actuators: right and left wheel speeds $[v_r, v_l]$, or linear and angular speeds $[v_l, v_w]$. To keep parity with the centralized model, our decentralized modeling considers two individual agents for learning v_l and a_w in parallel as is shown in Table 3.

Stage 3.2 — Identifying common and individual goals: the *Ball-Pushing* behavior can be separated into two sub-tasks, *ball-shooting* and *ball-goal-aligning*, which are performed respectively by $agent_{vl}$ and $agent_{a_w}$.

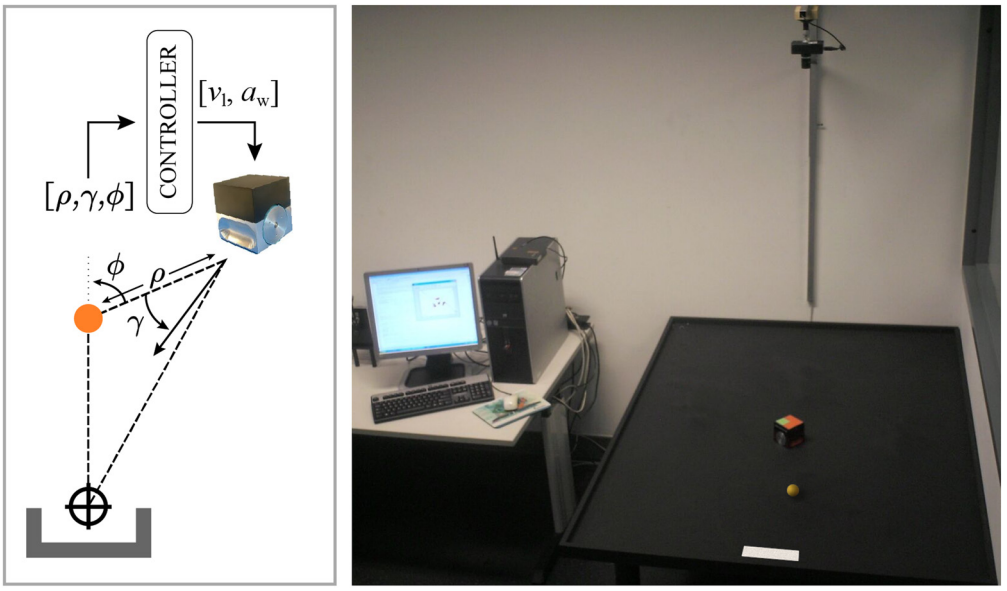

Fig. 5. Definition of variables for the Ball-Pushing problem (left), and, a picture of the experimental setup implemented for testing the Ball-Pushing behavior (right).

Table 3
Description of state and action spaces for the DRL modeling of the Ball-Pushing problem.

Joint state space: $S = [\rho, \gamma, \phi, v_w]^T$			
State variable	**Min.**	**Max.**	**N. cores**
ρ	0 mm	1000 mm	5
γ	−45 deg	45 deg	5
ϕ	−45 deg	45 deg	5
v_w	−10 deg/s	10 deg/s	5
Decentralized action space: $A = [v_l, a_w]$			
Agent	**Min.**	**Max.**	**N. actions**
v_l	0 mm/s	100 mm/s	7
a_w	−2 deg/s^2	2 deg/s^2	3
Centralized action space: $A = [v_l \cdot a_w]$ $N_T = N^{v_l} N^{a_w} = 5 \times 3 = 15$ actions			

Stage 3.3 — Defining the reward functions: a common reward function is considered for both CRL and DRL implementations, as is shown in Expression (10), where *max* features are normalization values taken from Table 3.

$$R(s) = \begin{cases} +1 & \text{if goal} \\ -(\rho/\rho_{max} + \gamma/\gamma_{max} + \phi/\phi_{max}) & \text{otherwise} \end{cases} \tag{10}$$

Stage 3.4 — Determining if the problem is fully decentralizable: the joint state vector $[\rho, \gamma, \phi, v_w]$ is identical to the one proposed for the centralized case.

Stage 3.5 — Completing RL single modelings: one of the main goals of this work is also validating DRL system benefits. And an interesting property of those kinds of systems is the flexibility to implement various algorithms or modelings independently by each individual agent. In this way, we have implemented a hybrid DRL scheme (DRL-Hybrid) with a Fuzzy Q-Learning (FQL) to learn v_l, in parallel with a SARSA(λ) algorithm to learn a_w. This is a good example for depicting Stage 3.5 of the proposed methodology. The continuous state but discrete actions RBF SARSA(λ) is adequate for learning the discrete angular acceleration. Meanwhile, the continuous state-action FQL algorithm is adequate for learning the continuous linear speed control action of the agent v_l. For simplicity, the DRL-Hybrid scheme is implemented with a greedy exploration policy, the same previously mentioned joint state vector, and 3 bins in the action space. It is also important to mention that any kind of MA coordination or algorithm (e.g., DRL-Lenient or DRL-CA) is implemented for this scheme.

In summary, we have the following implemented schemes for the Ball-Pushing problem: CRL, DRL-Ind, DRL-CAdec/CAinc/Lenient, and DRL-Hybrid. Please see Table 1 for the full list of acronyms. Other details about Stage 3.5 are detailed in the next two subsections. Implementation and environmental details have been already mentioned in the centralized modeling description, because most of them are common with the decentralized modeling.

5.2.3. Performance index

The evolution of the learning process is also evaluated by measuring and averaging 25 runs. The percentage of scored goals across the trained episodes is considered as the performance index: $\%of\ Scored\ Goals = scoredGoals/Episode$, where *scoredGoals* are the number of scored goals until the current training *Episode*. Final performance is also measured by running a thousand episodes again with the best policy (among 25) obtained per each scheme tested.

5.2.4. RL algorithm and optimized parameters

An RBF SARSA(λ) algorithm with softmax action selection is implemented for these experiments. The Boltzmann exploration temperature is decayed as: $\tau = \tau_0 \exp(-dec \cdot episode/maxEpisodes)$, where *episode* is the current episode index and $maxEpisodes = 1000$ trained episodes per run. Thus, the following parameters are optimized: the learning rate (α), the eligibility traces decay factor (λ), the Boltzmann exploration initial temperature (τ_0), and the exploration decay factor (*dec*). For the particular case of Lenient RL, the gain (κ) and decay factor (β) are optimized instead of τ_0 and *dec* respectively. Obtained values after optimizations are listed in Table 1. Additionally, the number of discrete actions for the linear velocity are optimized obtaining $N^{v_l} = 5$ for the CRL scheme, and $N^{v_l} = 7$ for the DRL-Ind. For all the experiments $\gamma = 0.99$.

5.2.5. Physical setup

An experimental setup is implemented in order to test learned policies onto a physical setup, which is shown in Fig. 5 (right). The Miabot Pro is connected wirelessly to a central computer close to the robot soccer platform which measures 1.5 m $\times$ 1 m. A web camera above the platform provides the positions and orientations of the robot, ball, and goal. The state observation is processed from the vision system, while the speed of the wheels is transmitted through Bluetooth from the computer. These speeds are computed from the Q tables by using a greedy search policy.

5.2.6. Results and analysis

Fig. 6 presents learning evolution plots and Table 4 shows the best policy final performances. All the DRL systems implemented improved the *%of ScoredGoals* of the centralized one as in the learning evolution traces (Fig. 6), as well as in the final performance test (Table 4). Except from the Incremental Cooperative Adaptive implementation, DRL-CAinc, the DRL implementations accelerated the learning time of the CRL scheme. Although DRL-CAinc achieves better performances than CRL after episode 500, the slower learning of the DRL-CAinc can be explained by taking the incremental cooperative adaptation effect into account, which updates the Q function conservatively during early episodes in which the agents do not have good policies. The hybrid SARSA+Fuzzy-Q-Learning decentralized implementation, DRL-Hybrid, shows the fastest asymptotic convergence, evidencing the feasibility of using decentralized systems with various algorithms and/or modelings for each individual agent, which means flexibility, property indicated in Section 2.3 and described in Stage 3.5. The Decremental Cooperative Adaptive implementation, DRL-CAdec, obtains the best final performance and the second fastest asymptotic convergence, followed closely by the DRL-Lenient scheme, and the independent and no coordinated DRL-Ind implementation. Note that coordinated DRL schemes (CA and Lenient) do not show considerable outperforming or accelerating with respect to the DRL-Ind implementation. This is an interesting point to analyze and discuss in the following sections, taking the previous results of the 3DMC problem into account, and the fact that this particular problem also uses two agents with full observability of the joint state space.

As was mentioned in Section 5.2, the number of discretized actions for the linear velocity was optimized, obtaining $N^{v_l} = 5$ for the CRL scheme, and $N^{v_l} = 7$ for the DRL-Ind. So, total discrete actions are: $N_T = N^{v_l} N^{a_w} = 15$ for the CRL scheme, and $N_T = 7 + 3$ for the DRL-Ind. Note that the DRL-Ind implementation allows a finer discretization than the CRL. For the CRL, increasing the number of actions of v_l from 5 to 7 implies increasing the joint action space from 15 to 21 actions, taking into account $N^{a_w} = 3$ (please check Table 3), which implies an exponential increase in the search space that could increase learning time, thus affecting the final performance since only 1000 episodes were trained. Although the DRL-Ind scheme uses more discrete actions for v_l, its search space is still smaller than the CRL combined one. This is one of the interesting properties of decentralized systems, which is validated by these optimization results. Since the agents are independent, separate modelings or configurations can be implemented per agent. Additionally, in order to perform a fair comparison of computing time, we have also carried out a second evaluation, implementing and testing a DRL system with $N^{v_l} = 5$ actions. Once again, we have measured simulation times and action-selection + Q-function-update times, obtaining 59.63 s for the CRL (12.47% of the global time), and 59.67 for the DRL scheme (15.11% of the global time). However the DRL five actions final performance was 68.97%, still higher than the 57.14% of its CRL counterpart, although lower than the 75.28% of the DRL with $N^{v_l} = 7$ actions.

The best CRL and DRL-Ind learned policies are transferred and tested in the experimental setup. The results from experiments with the physical robot are also presented in Table 4, in which performance is presented in percentages of success at scoring a goal within the seventy attempts. Cases where the mark of the robot was lost in the vision system were disregarded.

In Table 4 it is observed that DRL-Ind performs on average 11.43% better than CRL. Simulation and physical setup performances are similar, which validates the simulation experiments and results. Some experiments for centralized and decentralized RL were recorded and can be seen online at Leottau's video repository [50]. In this video actions are a bit abrupt as it can be seen, due to no smoothing or extrapolation of the discrete actions where carried out, policies were transferred directly from Q functions to the physical robot. Also, cases where the mark of the robot or some tracker was lost

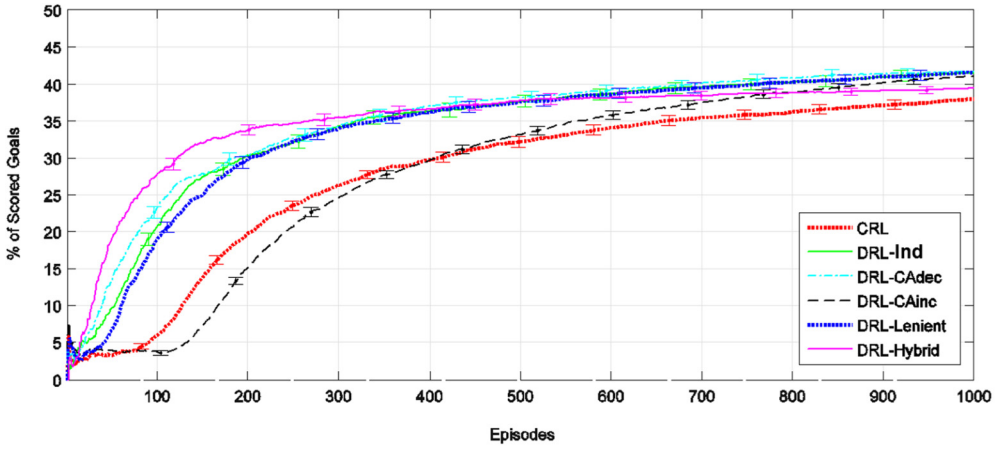

Fig. 6. Ball-Pushing learning evolution plots. Results are averaged across 25 learning runs and error bars show the standard error. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

Table 4
Ball-pushing best policy final performances for simulation and physical robot experiments (in which 100% is the optimal case).

Approach	Performance (%)
DRL-CAdec	76.69
DRL-Lenient	75.76
DRL-Ind	75.28
DRL-Hybrid	73.97
DRL-CAinc	71.24
CRL	62.15
DRL-Ind (physical robot)	68.57
CRL (physical robot)	57.14

in the vision system were disregarded. These aspects should be improved for future implementations, however, the purpose of this experiments is more focused on comparing CRL and DRL approaches, than on achieving an optimal performance.

5.3. Ball-dribbling

Ball-dribbling is a complex behavior during which a robot player attempts to maneuver the ball in a very controlled way, while moving toward a desired target. In the case of biped robots the complexity of this task is very high, because it must take into account the physical interaction between the ball, the robot's feet, and the ground. Thus, the action is highly dynamic, non-linear, and influenced by several sources of uncertainty. Fig. 7 (left) shows the RoboCup SPL soccer environment where the NAO humanoid robot [14] is used. As proposed by Leottau et al. [25], the description of this dribbling behavior uses the following variables: $[v_x, v_y, v_\theta]$, the velocity vector; γ, the robot-ball angle; ρ, the robot-ball distance; and, ϕ, the robot-ball-target complementary angle. These variables are shown in Fig. 7 (right), where the global coordinate system is O_G, the desired target ($\oplus$) is located in the middle of the opponent's goal, and a robot's egocentric reference system is indicated with the x_r axis pointing forwards.

5.3.1. Proposed decentralized modeling

Stage 3.1 — Determining if the problem is decentralizable: since the requested velocity vector of the biped walk engine is $[v_x, v_y, v_\theta]$, it is possible to decentralize this 3-Dimensional action space by using three individual agents, $Agent_x$, $Agent_y$, and $Agent_\theta$.

Stage 3.2 — Identifying common and individual goals: our expected common goal is to walk fast toward the desired target while keeping possession of the ball. That means: to maintain $\rho < \rho_{th}$; to minimize $\gamma, \phi, v_y, v_\theta$; and to maximize v_x. In this way, this Ball-Dribbling behavior can be separated into three sub-tasks or individual goals, which have to be executed in parallel: *ball-turning*, which keeps the robot tracking the ball-angle ($\gamma = 0$); *alignment*, which keeps the robot aligned to the ball-target line ($\phi = 0$); and *ball-pushing*, whose objective is for the robot to walk as fast as possible and hit the ball in order to change its speed, but without losing possession of it. So, the proposed control actions are $[v_x, v_y, v_\theta]$ respectively involved with *ball-pushing*, *alignment*, and *ball-turning*.

Stage 3.3 — Defining the reward functions: the proposed dribbling modeling has three well-defined individual goals, *ball-pushing*, *alignment*, and *ball-turning*. Thus, individual rewards are proposed for each agent as:

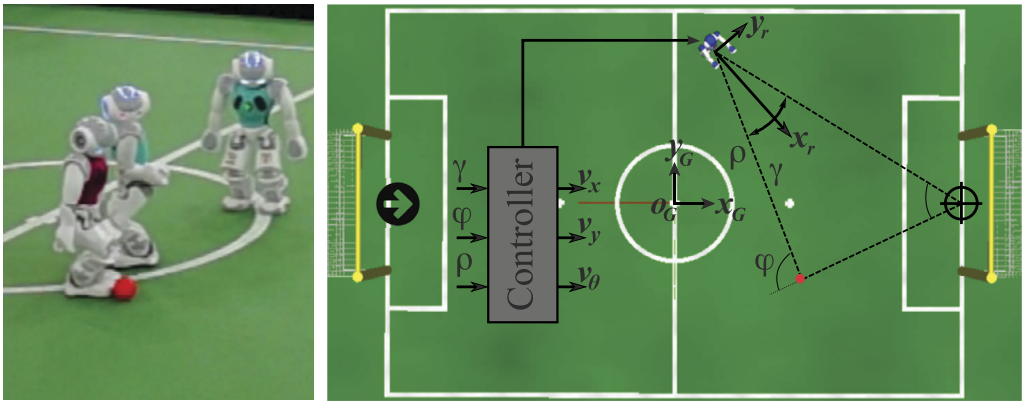

Fig. 7. A picture of the NAO robot dribbling during a RoboCup SPL game (left), and definition of variables for dribbling modeling (right).

Table 5
Description of state and action spaces for the DRL modeling of the Ball-Dribbling problem.

Joint state space: $S = [\rho, \gamma, \phi]^T$			
State variable	**Min.**	**Max.**	**N. cores**
ρ	0 mm	800 mm	13
γ	$-60°$	$60°$	11
ϕ	$-60°$	$60°$	11
Action space: $A = [v_x, v_y, v_\theta]$			
Agent	**Min.**	**Max.**	**N. actions**
v_x	0 mm/s	100 mm/s	17
v_y	-50 mm/s	50 mm/s	17
v_θ	$-45°/s^2$	$45°/s^2$	17

$$r^x = \begin{cases} 1 & \text{if } \rho < \rho_{th} \wedge \gamma < \gamma_{th} \wedge \phi < \phi_{th} \wedge v_x \geq v_{x.max'} \\ -1 & \text{otherwise} \end{cases}$$
$$r^y = \begin{cases} 1 & \text{if } \gamma < \gamma_{th}/3 \wedge \phi < \phi_{th}/3 \\ -1 & \text{otherwise} \end{cases} \quad (11)$$
$$r^\theta = \begin{cases} 1 & \text{if } \gamma < \gamma_{th}/3 \wedge \phi < \phi_{th}/3 \\ -1 & \text{otherwise} \end{cases}$$

where $[\rho_{th}, \gamma_{th}, \phi_{th}]$ are desired thresholds at which the ball is considered to be controlled, while $v_{x.max'}$ reinforces walking forward at maximum speed. Fault-state constraints are set as: $[\rho_{th}, \gamma_{th}, \phi_{th}] = [250 \text{ mm}, 15°, 15°]$, and $v_{x.max'} = 0.9 v_{x.max}$. This is a good example for depicting how and why to define individual rewards; for instance, since only $Agent_x$ involves v_x for the *ball-pushing* sub-task, $Agent_y$, and $Agent_\theta$ reward functions do not include this variable. Since *alignment*, and *ball-turning* strongly involve γ and ϕ, $Agent_y$ and $Agent_\theta$ rewards consider more accurate thresholds for these angles, $\gamma_{th}/3$, $\phi_{th}/3$ and ρ is also not considered.

Stage 3.4 — Determining if the problem is fully decentralizable: since the three state variables, $[\rho, \gamma, \phi]$ of the joint vector state are required, this problem is not considered to be fully decentralizable. So, the proposed modeling for learning the 3-Dimensional velocity vector from the joint observed state is detailed in Table 5.

5.3.2. Centralized modeling

Since 17 discrete actions per agent are implemented for the DRL system, if an equivalent CRL system were implemented, that centralized agent would search in an action space of $17^3 = 4913$ possible actions, which would be enormous for most of the RL algorithms. Even though we tried to reduce the number of discrete actions, the performance decreased dramatically. Finally, the only way to achieve asymptotic convergence was using a noiseless model in which observations were taken from the ground truth system. Thus, this CRL implementation is only for academic and comparison purposes. Discrete actions must have been reduced up to five per action dimension, i.e. a $5^3 = 125$ combined action space. The same joint state vector was used and the global reward function is similar to r^x in (11), but using $\gamma_{th}/3$ and $\phi_{th}/3$.

Stage 3.5 — Completing RL single modelings: the Ball-Dribbling DRL procedure is carried out episodically. After a reset, the robot is set in the center of its own goal (black right arrow in Fig. 7 (right)), the ball is placed ρ_{th} *mm* in front of the robot, and the desired target is defined in the center of the opponent's goal. The terminal state is reached if the robot loses the ball, if the robot leaves the field, or if the robot crosses the goal line and reaches the target, which is the expected terminal

state. The training field is 6×4 meters. In order to compare our proposed methods with similar state-of-the-art works, three additional schemes, previously reported in the literature, are included:

- DRL + Transfer, a DRL system accelerated by using the Nearby Action-State Sharing (NASh) knowledge transfer approach proposed by Leottau and Ruiz-del-Solar [26]. NASh is introduced for transferring knowledge from continuous action spaces, when no information different from the suggested action in an observed state is available from the source of knowledge. In the early training episodes, NASh transfers actions suggested by the source of knowledge but progressively explores its surroundings looking for better nearby actions for the next layer.
- RL-FLC method introduced by Leottau et al. [25], which proposes a methodology for modeling dribbling behavior by splitting it in two sub problems: *alignment*, which is achieved by using an off-line tuned fuzzy controller, and *ball-pushing*, which is learned by using an RL based controller reducing the state vector only to ρ. These strategies reduce the complexity of the problem making it more tractable and achievable for learning with physical robots. The RL-FLC approach was the former dribbling engine used by the UChile Robotics Team [55] in the RoboCup [51] Standard Platform League (SPL) soccer competition.
- eRL-FLC proposed by Leottau et al. [27], is an enhanced version of the RL-FLC which learns the *ball-pushing* sub-task mapping the whole state space $[\rho, \gamma, \phi]$ by using a Layered RL approach. It is designed to improve ball control because the former RL-FLC approach assumes the ideal case in which the target, ball, and robot are always aligned, ignoring $[\gamma, \phi]$ angles, which is not the case during a real game situation. However, as in RL-FLC, the *alignment* sub-task must still be learned off-line, resigning optimal performances instead of reducing modeling complexity.

In summary, for implemented schemes for the Ball-Dribbling problem, we have: DRL-Ind, DRL-CAdec/CAinc/Lenient, DRL+Transfer, CRL, RL-FLC, and eRL-FLC. Please see Table 1 for the full list of acronyms. Other details about Stage 3.5 are detailed in the next two subsections.

5.3.3. Performance indices

The evolution of the learning process is evaluated by measuring and averaging 25 runs. The following performance indices are considered to measure *dribbling-speed* and *ball-control* respectively:

- *% of maximum forward speed* ($\%S_{Fmax}$): given S_{Favg}, the average dribbling forward speed of the robot, and S_{Fmax}, the maximum forward speed: $\%S_{Fmax} = S_{Favg}/S_{Fmax}$. $\%S_{Fmax} = 100\%$ is the best performance.
- *% of time in fault-state* ($\%T_{FS}$): the accumulated time in *fault-state* t_{FS} during the whole episode time t_{DP}. The *fault-state* is defined as the state when the robot loses possession of the ball, i.e. $\rho > \rho_{th} \ \vee \ |\gamma| > \gamma_{th} \ \vee \ |\phi| > \phi_{th}$, then: $\%T_{FS} = t_{FS}/t_{DP}$. $\%T_{FS} = 0$ is the best performance.
- *Global Fitness (F):* this index is introduced for the sole purpose of evaluating and comparing both performance indices together. The global fitness is computed as follows: $F = 1/2[(100\%S_{Fmax}) + \%T_{FS}]$, where $F = 0$ is the optimal but non-reachable policy.

5.3.4. RL algorithm and optimized parameters

A SARSA(λ) algorithm with softmax action selection is implemented for these experiments. The Boltzmann exploration temperature is decayed as: $\tau = \tau_0 \exp(\text{-}dec \cdot episode/maxEpisodes)$, where *episode* is the current episode index and $maxEpisodes = 2000$ trained episodes per run. As a result, the following parameters are optimized: learning rate (α), Boltzmann exploration initial temperature (τ_0), and exploration decay factor (dec). For the particular case of Lenient RL, gain (κ) and decay factor (β) are optimized instead of τ_0 and dec respectively. This can be considered as the last stage of the methodology, *Stage 3.5 Completing RL single modelings*. For all the experiments $\gamma = 0.99$.

5.3.5. Results and analysis

Fig. 8 shows learning evolution plots for: an independent decentralized learners scheme, DRL-Ind; a Decremental Cooperative Adaptive scheme, DRL-CAdec; a lenient DRL implementation; and, a DRL system accelerated with transfer knowledge, DRL+Transfer, and a centralized scheme, CRL. Plots for the Incremental Cooperative Adaptive, DRL-CAinc scheme are not included because of their poor performance and high standard error bars. Table 6 shows averaged final performances. Although CRL implementation was modeled with only 5 actions per dimension, the DRL-ind scheme which uses 17 actions per dimension has been more than 22% faster. Besides, the CRL has used a noiseless model with ground truth observations, even so DRL outperforms it by almost 12% using a more realistic model. The DRL+Transfer implementation uses a source of knowledge with an initial performance at about 25% (see [26]), achieving a final performance near 16% after the RL transfer procedure. DRL-Lenient and DRL-CAdec approaches are able to reach a similar final performance, approximately 18% and 20% learning from scratch without any kind of previous knowledge. The Lenient approach presents the best results, the best final performance, and the fastest asymptotic convergence among the implemented methods with no transfer knowledge. The DRL-CAdec outperforms the DRL-Ind scheme, and also takes 201 less episodes to reach the defined time to threshold (35%). Plots for forward speed and fault performance indexes are also included in order to follow the same results format as Leottau and Ruiz-del-Solar [26], in which this dribbling problem was originally proposed based on a DRL modeling. Note

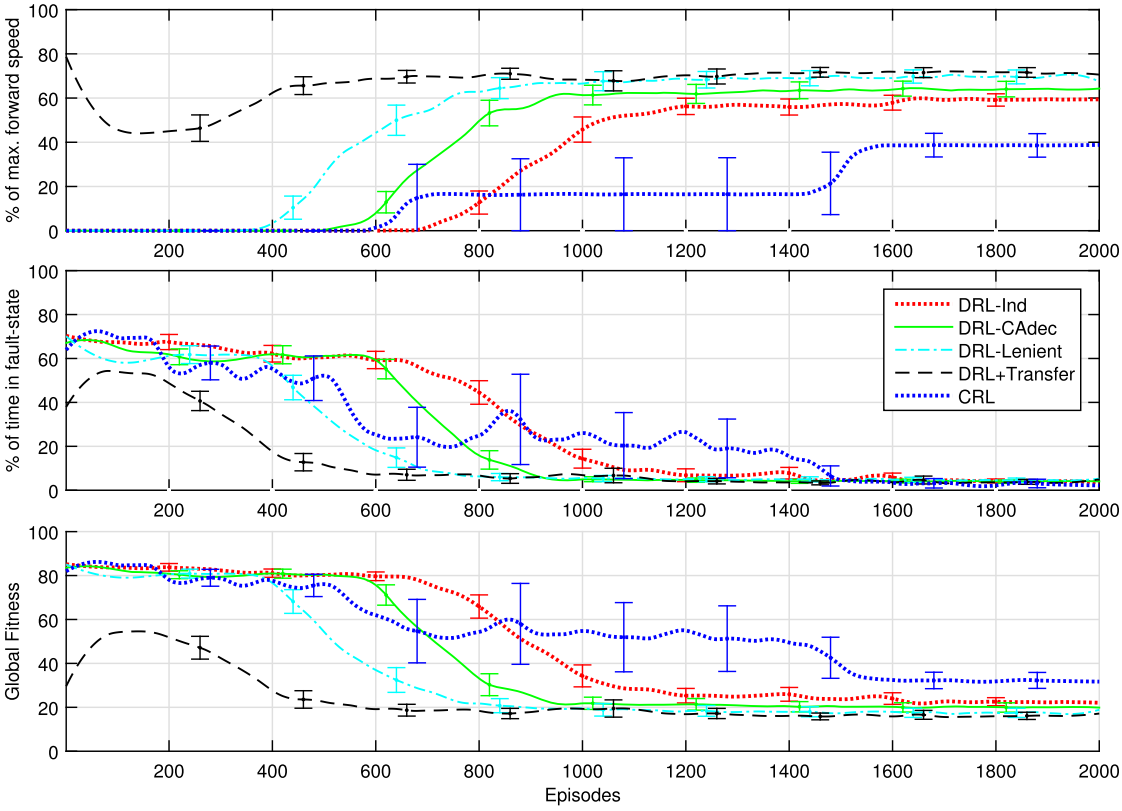

Fig. 8. Ball-dribbling learning evolution plots.

Table 6
Ball-Dribbling performances (in which lower %s are better performances).

Approach	**Performance (%)**	**Time to Th.** (35%)
DRL + Transfer	16.36	356
DRL-Lenient	17.78	585
DRL-CAdec	20.44	771
DRL-Ind	23.29	972
eRL-FLC	27.67	61
RL-FLC	34.40	47
DRL-CAinc	77.43	†
CRL	34.93	1419

† It did not achieve asymptotic convergence before the 2000 trained episodes, because of that its learning evolution plot is neither included in Fig. 8.

that the main benefit of MAS based algorithms (Lenient and CAdec) versus the DRL-Ind scheme is to achieve a higher forward speed, keeping a low rate of faults.

The effectiveness and benefits of the RL-FLC and eRL-FLC approaches have been pointed out by Leottau et al. [27]. However, significant human effort and knowledge from the controller designer are required for implementing all the proposed stages. DRL approaches are able to learn the whole Ball-Dribbling behavior almost autonomously, achieving best performances compared to those of the RL-FLC and eRL-FLC which require more human effort and previous knowledge. An advantage of the RL-FLC and eRL-FLC methods is the considerably lower RL training time, with regard to all the DRL systems (please see time to thresholds in Table 6). The DRL-Lenient and DRL-CA schemes proposed in this work are able to reduce the learning time down to 585 and 771 episodes respectively, opening the door to making future implementations for learning similar behaviors achievable by physical robots.

Some videos showing the learned policies for dribbling can be seen online at Leottau's video repository [24]. Currently the learned policy is transferred directly to the physical robots, thus, the final performance is dependent on how realistic the simulation platform is.

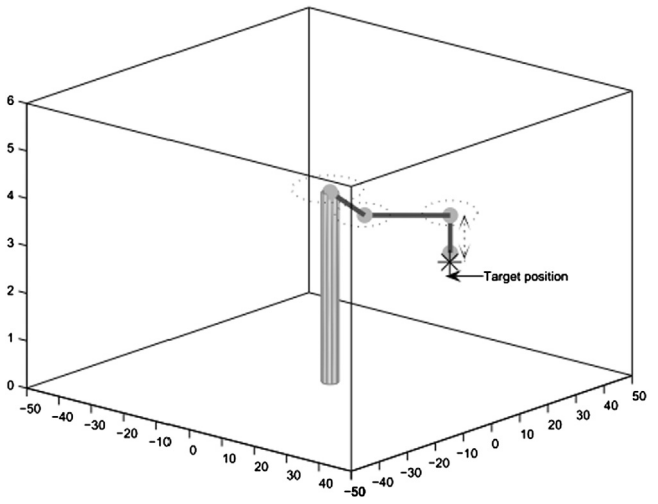

Fig. 9. The SCARA robotic manipulator (figure adopted from Martin and De Lope [31]).

5.4. SCARA real-time trajectory generation

The Selective Compliance Articulated Robot Arm (SCARA) is used extensively in the industry for assembly tasks and small parts insertion, among the other uses. It has well-known properties and there is sufficient literature [29,10]. This problem has been selected because it is one of the first DRL applications reported by Martin and De Lope [31]. Its simulation implementation is available online at Martin's repository [30] and it can be used as a test-bed for DRL systems.

A simulated three dimensional robotic manipulator with four joints, in which the system tried to reach an objective configuration in a 3D space, was used, while being able to generate an approaching real-time trajectory when the system was completely trained. A diagram of the physical model of the SCARA-RTG problem is shown in Fig. 9. The Denavit–Hartenberg parameters, the direct kinematic matrix, and more implementation details can be checked in the paper of Martin and De Lope [31], and the source files can be downloaded from Martin's repository [30].

5.4.1. Decentralized modeling

Stage 3.1 — Determining if the problem is decentralizable: since the SCARA arm has four joints, we can identify a 4-dimensional action space, and separate the problem into four individual agents: $Agent_1, \cdots, Agent_4$, or $Agent_m$, with $m = 1, .., 4$. Five actions are implemented per agent, among which the four action spaces are identical, but act independently: $A^1 = A^2 = A^3 = A^4 = [-0.02; -0.01; 0.0; 0.01; 0.02]$. Selected action $[a^1, \cdots, a^4]$ modifies the current angle in radians of each joint of the arm $[\theta_1, \cdots, \theta_4]$. Thus, $\theta(t+1)_m = \theta_m + a^m$.

Since this problem is modeled with four agents and five discrete actions per agent, a centralized scheme is not implemented for this experiment because an action-space of $5^4 = 625$ discrete actions is computationally expensive to our current resources and purposes.

Stages 3.2 and 3.3 — Identifying individual goals and defining reward functions: the common goal consists of reaching a continuously changing goal position of the robot end-effector by means of a random procedure. That way, a global reward function is implemented as Expression (12), where $eDist$ is the Euclidean distance between the current end-effector position and the goal position, and θ_2 is the joint angle $m = 2$ in degrees. In Martin's repository [30], this Euclidean distance based and continuous reward function is detailed and its effectively is validated.

$$R(s) = \begin{cases} 10^8/(1+eDist^2) & \text{if } eDist \leq 1 \;\&\; penal = 1 \\ -penal \cdot eDist^5/10^4 & \text{otherwise} \end{cases}$$
$$penal = \begin{cases} 10 + 0.1\theta_2 & \text{if } \theta_2 < 0 \\ 1 & \text{otherwise} \end{cases} \tag{12}$$

Stage 3.4 — Determining if the problem is fully decentralizable: three state variables compose the *joint state vector* $S = [e_x, e_y, e_z]$, the error between the current end-effector position with respect to the 3-Dimensional goal position point $[x_g, y_g, z_g]$. Each state variable considers three values $[-1, 0, 1]$, depending if the error is negative, near to zero, or positive.

Stage 3.5 — Completing RL single modelings the learning procedure is defined as follows: goal positions are defined in such a way that they are always reachable for the robot; thus, the learning process needs to develop an internal model of the inverse kinematics of the robot which is not directly injected by the designer; through the different trials, a model of the robot inverse kinematics is learned by the system; when a goal position is generated, the robot tries to reach it; each trial can finish as a success episode, i.e. the robot reaches the target at a previously determined time, or as a failed episode, i.e.

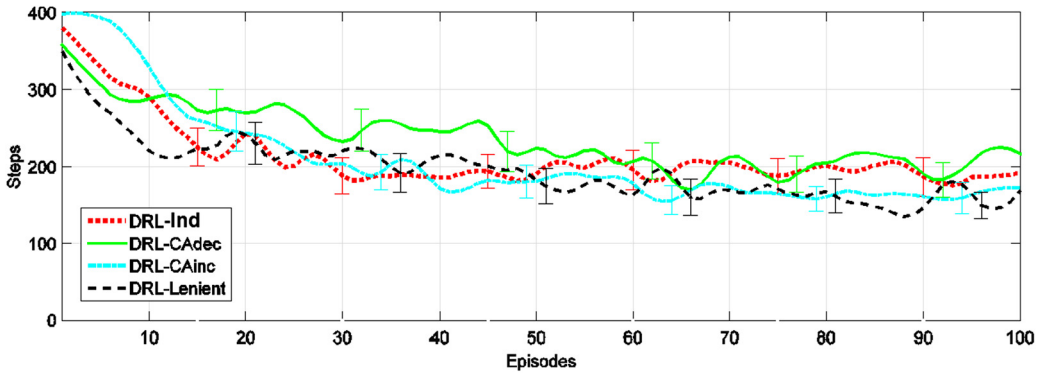

Fig. 10. SCARA-RTG learning evolution plots.

Table 7
SCARA-RTG performances (these improve toward zero).

Approach	Performance (steps)	Standard error
DRL-Lenient	159.05	18.63
DRL-CAinc	170.70	18.82
DRL-Ind	184.77	23.66
DRL-CAdec	209.57	23.06

the robot is not able to adopt a configuration to reach the goal position before 400 time steps; in both cases the system parameters are updated using the previously defined method and a new goal position is generated randomly.

In summary, we have the following implemented schemes for the SCARA-RTG problem: DRL-Ind, DRL-CAdec/CAinc/Lenient. Please see Table 1 for the full list of acronyms. Other details about Stage 3.5 are detailed in the next two subsections.

5.4.2. Performance index

Time steps are considered as the performance index, where 400 is the maximum and worst case, and zero is the best but non-reachable performance.

5.4.3. RL algorithm and optimized parameters

As Martin and De Lope [31] report, a SARSA tabular algorithm with ϵ-greedy is implemented for these experiments. The following parameters are optimized: learning rate (α) and exploration (ϵ), which is multiplied by 0.99 at the end of each learning episode. For the particular case of Lenient RL, gain (κ) and decay factor (β) are also optimized. For all the experiments $\gamma = 1$.

5.4.4. Results and analysis

Fig. 10 shows learning evolution plots and Table 7 shows averaged final performances. The DRL-Lenient scheme shows the fastest asymptotic convergence and the best performance, followed closely by the Incremental Cooperative Adaptive DRL-CAinc implementation. Those schemes respectively outperform about 25 and 14 steps with respect to the original implementation presented by Martin and De Lope [31]. Note that DRL-Lenient reaches a performance of ≈ 230 in about 10 episodes, and the independent and non-coordinated scheme, DRL-Ind in 16 episodes. Further, due to the leniency effect during early episodes, which tries to avoid uncoordinated and ambiguous information among the interaction of the four agents, DRL-Lenient keeps improving its performance until the final episode. There is a similar lenient effect in the case of DRL-CAinc; reaches the 230 performance threshold in about 25 episodes, on average 15 and 9 episodes slower than DRL-Lenient and DRL-Ind respectively. However, DRL-CAinc shows a comparable performance with respect to the Lenient after that and also shows the tendency to keep improving its performance during the learning procedure.

5.5. Discussion

Our first goal is demonstrating empirically that an independent DRL system is able to achieve similar learning times and comparable or slightly lower performances (because of lack of coordination) compared to CRL schemes. Thus, a trade-off between DRL benefits indicated in Section 2.3 and performance may be expected. Experiments of 3DMC and Ball-Pushing were the first presented. Results for those two problems have evidenced surprisingly that DRL-Ind implementations show better performances and faster learning times than their CRL counterparts. Only in the case of 3DMC, the DRL-ObsL-Ind approach shows a lower final performance, which was expected when taking into account that complex scenario with lack of coordination and limited observability. Nevertheless, the DRL-ObsL-Ind approach should be compared with a CRL-ObsL implementation for an equitable comparison; however, this CRL implementation with limited observability is simply

Table 8
Summary of the best methods implemented.

Problem	Best methods
3DMC Full obs.	DRL-CAdec & DRL-Lenient
3DMC Limited obs.	DRL-CAinc & DRL-Lenient
Ball-pushing	DRL-CAdec & DRL-Lenient
Ball-dribbling	DRL-Lenient & DRL-CAdec
SCARA-RTG	DRL-Lenient & DRL-CAinc

non-feasible because the CRL scheme does not support incomplete observations. All the same, as is demonstrated in Subsection 5.1, our proposed MAL algorithms (Lenient and CA) are able to resolve that issue. Furthermore, implementations with these MAL algorithms also showed better or comparable averaged performances and faster learning times compared with CRL implementations. Only in the ball-pushing case, DRL-CAinc shows slower convergence than CRL. This result, however, is because of the nature of CA: DRL-CAinc and DRL-CAdec can be mutually exclusive in certain cases, as was introduced in Section 4.3 and discussed below. In short, thirteen different DRL implementations have been implemented and compared with their CRL counterparts for these two initially discussed problems: eight for the 3DMC and five for the Ball-Pushing. Eleven of those thirteen DRL implementations have evidenced better or comparable averaged performances as well as faster asymptotic convergences when compared to their CRL counterparts, one has shown faster convergence but lower final performance, and one has shown better performance but slightly slower asymptotic convergence.

Two more MA algorithms have been presented, Lenient and Cooperative Adaptive Learning Rate, which have been considered in order to include a coordination mechanism among decentralized agents learning in parallel. The effects of these algorithms are mentioned briefly below in general terms:

- For DRL-Lenient, leniency helps the agents to explore and find a partially coordinated joint policy before starting to update the action-value function. Since no communication among the agents is performed, and they modify their action-selection mechanisms, a coordinated policy is achieved indirectly. The agents visit relevant states repetitively, searching for the best individual actions, which accomplishes a desired joint behavior; meanwhile, action-value functions are updated gradually once the agents' visit states are known.
- For DRL-CAinc/dec, a measure of the current quality of each individually performed action is communicated among the agents; then, a joint adaptive learning rate is computed according to the "worst" agent. If the CAinc approach is performed, a similar lenient effect occurs, and each individual action-value function is updated with that cooperative adaptive learning rate, thereby increasing the learning rate while a joint policy is improved during the learning process. Otherwise, if CAdec is used, the agents try to collect information during the early learning process, thereby decreasing the learning rate while a joint policy is learned.

The benefits of the Lenient and CA algorithms are more noticeable in those implementations of the 3DMC with limited observability, in which the DRL-ObsL-Ind scheme without coordination did not achieve a stable final performance. This particular case is highly complex because the actions of each agent affect the joint environment and next state observation for both agents directly, and not even free or indirect coordination occurs. However, Lenient and CA schemes were able to resolve that issue. As was mentioned in Section 5.1, this 3DMC problem with ObsL presents different state spaces, decentralized action spaces, and individual reward functions. So, by using our proposed MA algorithms, with their indirect coordination and communication among agents of the CA approaches, an equilibria can be found for every state which is visited along a successful path enough times to achieve the mountaintop.

Lenient and CA algorithms have evidenced the best averaged final performances for the four tested problems. These methods outperformed their DRL-Ind counterparts implemented without coordination in all the problems tested. The two best averaged performances per problem are listed in Table 8. DRL-Lenient is the most recurrent winning approach, appearing in all five cases. Lenient benefits are particularly remarkable in the Ball-Dribbling and SCARA-RTG problems, where it achieves both the best performance and the fastest asymptotic convergence. According to the results, the benefits of the proposed MAS methods are more noticeable as the problem complexity increases — such as occurs in the 3DMC ObsL and Ball-Dribbling cases — in which a CRL scheme was intractable according to our available computational resources.

Note in Table 8 that DRL-CAdec and DRL-CAinc never appear together as the best approaches. This verifies that DRL-CAinc and DRL-CAdec can be mutually exclusive for certain cases due to their inverse variable learning rates policy. For instance, DRL-ObsF-CAdec is the best, and DRL-ObsF-CAinc is the worst, in the 3DMC with full observability and Ball-Dribbling cases; and DRL-CAinc is the second best, and DRL-CAdec is the worst, in the SCARA-RTG case. As a preliminary and empirical hypothesis about DRL-CAxxx methods, it can be said that the DRL-CAinc method potentiates its benefits on learning problems implemented with the ϵ-greedy action-selection mechanism, but shows poor performances on problems implemented with softmax action selection. On the other hand, the DRL-CAdec method potentiates its benefits on learning systems implemented with a softmax action-selection strategy. Of course this is just an empirical conclusion which must be validated with more problems in future studies. Also, note in Figs. 4–10 that DRL-CAxxx approaches do not usually obtain the fastest asymptotic convergences, and as such it is possible to conclude that accelerating learning is not a strength of those methods.

For the sake of simplicity, we have used a unique set of RL parameters for each of the DRL implemented problems. If we had optimized individual sets of parameters per each individual agent, results may have been outperformed for those DRL schemes in which the agents are heterogeneous such as ball-pushing and dribbling.

6. Conclusions and future direction

In this article we addressed the Decentralized Reinforcement Learning (DRL) of individual behaviors of those problems in which multi-dimensional action spaces are involved. First, we have promoted and proposed a five-stages methodology to model and implement a DRL system in which basic concepts, definitions, and practical implementation issues were presented. Second, three Multi-Agent Learning (MAL) algorithms were detailed: independent DRL scheme (DRL-Ind), which does not consider any kind of cooperation or coordination among the agents; the Cooperative Adaptive Learning Rate (DRL-CA) approach, an original contribution which adapts the global learning rate on-line based on a simple estimation of the partial quality of the policy performed by the "weakest" agent; and DRL-Lenient, in which the value function is optimistically updated whenever the last performed action increases the current utility function, or it is leniently updated if the agent has explored that action sufficiently. Particularly DRL-CA and DRL-Lenient add coordination mechanisms in DRL-Ind systems.

The proposed DRL methodology and the three considered MAL algorithms were validated with an extensive empirical study on four different problems; two of them are well-known problems: the Three-Dimensional Mountain Car (3DMC), and a SCARA Real-Time Trajectory Generation (SCARA-RTG); and two correspond to noisy and stochastic real-world mobile robot problems: *Ball-Dribbling* in soccer performed with an omnidirectional biped robot, and the *Ball-Pushing* behavior performed with a differential robot. Results for 3DMC and Ball-Pushing problems evidenced that DRL implementations show better performances and faster learning times than their CRL (centralized RL) counterparts, even with less computational resources, and non-direct coordination mechanisms. On the other hand, DRL-Lenient and DRL-CA MAL algorithms showed the best final performances for the four tested problems, outperforming their DRL-Ind counterparts in all the problems.

Finally, note that the benefits of the proposed MAL methods were more remarkable as the problem complexity increased, such as occurs in the 3DMC ObsL and Ball-Dribbling cases, in which a CRL scheme is infeasible. These results empirically demonstrate that benefits of MAS are also applicable to more complex and real world problems like robotic platforms when using a DRL modeling. Furthermore, the results show DRL as a promising approach to develop applications with higher dimensional action spaces where a CRL scheme could not be easily implementable, such as behavior learning for snake robots, multi-link robotic arms, omnidirectional mobile robots, multi-rotor aerial vehicles, etc.

As part of our ongoing research agenda, we plan to combine the benefits of both DRL-CAdec and DRL-CAinc, in order to develop a unique and improved cooperative adaptive method. As a related idea, we are interested in developing a DRL-CA version in which individual adaptive learning rates per action-state pair are available, as well as a full adaptive DRL-CA version where exploration is also dependent on that adaptive parameter.

There are a number of possible directions for future work. For now DRL-Lenient and DRL-CA have been implemented based on temporal-difference and discrete action RL methods, and so extending these two methods to model-based and actor-critic algorithms remains an area for future work. Another topic for future work is comparing partial observable MDP MAL algorithms to our DRL-CAdec and DRL-CAinc methods which have shown good results under limited observation conditions. Finally, an interesting research direction is that of exploring possibilities for automated sub-task detection and decomposition. Additionally, since in DRL an agent can be decomposed into several separate agents, real-time communication and observation among those individual agents is not an issue unlike many of the MAS. Thus, sharing information can be the basis for a research line in the field of distributed artificial intelligence, that has not been sufficiently explored yet, in which increasingly sophisticated DRL algorithms can be developed, taking advantage of DRL systems' properties.

Acknowledgements

David L. Leottau was funded by CONICYT under grant CONICYT-PCHA/Doctorado Nacional/2013-63130183. This research was also partially funded by FONDECYT Project 1161500 and by the European Regional Development Fund under the project Robotics 4 Industry 4.0 (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000470). The authors thank to Aashish Vatsyayan for the Ball-Pushing experiments with the physical setup, and Jose Antonio Martin for sharing his RL source code.

Appendix A. Optimization procedure

As it was mentioned, RL parameters like learning rate, eligibility traces, exploration factor, number of discrete actions, number of cores, are optimized by using a customized version of the hill-climbing method. This is a very important step in order to guarantee that every scheme tested uses the best parameter settings. In this way our comparisons and evaluations are carried out based on the best performance potentially achievable by each method, according to our optimization results. Before each set of optimizations, we try to achieve a good set of parameters by hand-tuning, such as seed, and then we determined the quantity of learning episodes empirically procuring asymptotic convergence for 2/3 of the total trained episodes.

The relative simplicity and fast convergence of hill climbing algorithm make it one of the most popular algorithms for finding the best set of parameters in RL [3,5,34,46]. However, since only local optima are guaranteed, we have implemented

some variants to cure that without evaluating too much extra trials. In this way three ideas are included: to evaluate more than one neighbor per parameter dimension; the option of evaluating one neighbor per dimension or exploring the same dimension until finding the best evaluation; and to store every evaluated set of parameters in order to avoid repeated trials. The pseudo-code is detailed in Algorithm 4, where *paramListV* is a structure which stores every parameter combination and its respective evaluation value. We have used $neighbors = 4$ and *oneDimPerTry* enabled for all the experiments in this work. The source code is also shared on-line at Leottau's code repository [23].

Algorithm 4 Customized hill climbing algorithm.

Parameters:
x_0 ▷ Initial parameter
D ▷ Number of dimensions of the parameters space
$neighbors$ ▷ Number of neighbors to explore
$timeLimit$ ▷ Maximum time available
$maxIter$ ▷ Maximum number of iterations desired
$[paramMin^1, \cdots, paramMin^D]$ ▷ Lower boundary of the parameter space
$[paramStep^1, \cdots, paramStep^D]$ ▷ Step size of the parameter space
$[paramMax^1, \cdots, paramMax^D]$ ▷ Upper boundary of the parameter space
$goal$ ▷ Desired value after optimization
$oneDimPerTry$ ▷ Enables exploration in one dimension until find the best evaluation
$paramListV \leftarrow BuildParamList(paramMin, paramStep, paramMax)$
Initialize:
$iter \leftarrow 1,\ fval \leftarrow \infty,\ paramListV \leftarrow \infty \cdot paramListV$
$p_0 \leftarrow GetParamIndex(x_0)$
$paramListV(p_0) \leftarrow GetEval(x_0)$
$p \leftarrow p_0$
procedure until ($fval \leq goal$ OR $iteration \geq maxIter$ OR
$elapsedTime \geq timeLimit$ or $\forall ParamListV < \infty$)
for all Parameter Dimension $d \in D$ **do**
  $ymin \leftarrow -\infty$
  while $ParamListV(p_0) > ymin$ **do**
    $p_0 \leftarrow p$
    $x_0 \leftarrow GetParam(p_0)$
    $x \leftarrow x_0$
    for neighbor $n = 1, \cdots, neigbors$ **do**
      $x^d \leftarrow \max(x_0^d - n \cdot paramStep^d, paramMin^d)$
      $p \leftarrow GetParamIndex(x)$
      if $ParamListV(p) == \infty$ **then**
        $ParamListV(p) \leftarrow GetEval(x)$
      end if
    end for
    for neighbor $n = 1, \cdots, neigbors$ **do**
      $x^d \leftarrow \min(x_0^d + n \cdot paramStep^d, paramMax^d)$
      $p \leftarrow GetParamIndex(x)$
      if $ParamListV(p) == \infty$ **then**
        $ParamListV(p) \leftarrow GetEval(x)$
      end if
    end for
    if $NOToneDomentionPerTry$ **then**
      $p_0 \leftarrow p$
    end if
  end while
end for
if $fval \leq ymin$ **then**
  BREAK
end if
  $fval \leftarrow ymin$
end procedure return($fval, x = GetParam(p)$)

References

[1] D. Bloembergen, M. Kaisers, K. Tuyls, Lenient frequency adjusted Q-learning, in: 22nd Belgium–Netherlands Conf. Artif. Intel., 2010, pp. 19–26.
[2] H. Bou-Ammar, H. Voos, W. Ertel, Controller design for quadrotor UAVs using reinforcement learning, in: 2010 IEEE Int. Conf. Control Appl., 2010, pp. 2130–2135.
[3] M. Bowling, M. Veloso, Multiagent learning using a variable learning rate, Artif. Intell. 136 (2) (2002) 215–250.
[4] M. Brafman, I. Ronen, M. Tennenholtz, R-max – a general polynomial time algorithm for near-optimal reinforcement learning, J. Mach. Learn. Res. 3 (1) (2003) 213–231.
[5] L. Busoniu, R. Babuska, B. De-Schutter, A comprehensive survey of multiagent reinforcement learning, IEEE Trans. Syst. Man Cybern., Part C, Appl. Rev. 38 (2) (2008) 156–172.
[6] L. Busoniu, R. Babuska, B.D. Schutter, D. Ernst, Reinforcement Learning and Dynamic Programming Using Function Approximators, CRC Press, Boca Raton, Florida, 2010.

[7] L. Busoniu, B. De-Schutter, R. Babuska, Decentralized reinforcement learning control of a robotic manipulator, in: Ninth Int. Conf. Control. Autom. Robot Vision, ICARCV, Singapore, 2006, pp. 1–6.
[8] C. Claus, C. Boutilier, The dynamics of reinforcement learning in cooperative multiagent systems, in: Proc. Fifteenth Natl. Conf. Artif. Intell. Appl. Artif. Intell., IAAI '98, Madison, Wisconsin, USA, 1998, pp. 746–752.
[9] R.H. Crites, A.G. Barto, Improving elevator performance using reinforcement learning, in: Adv. Neural Inf. Process. Syst., vol. 8, NIPS, Denver, USA, 1995, pp. 1017–1023.
[10] M.T. Das, L. Canan Dülger, Mathematical modelling, simulation and experimental verification of a SCARA robot, Simul. Model. Pract. Theory 13 (3) (2005) 257–271.
[11] U. Dziomin, A. Kabysh, V. Golovko, R. Stetter, A multi-agent reinforcement learning approach for the efficient control of mobile robot, in: IEEE Conf. Intell. Data Acquis. Adv. Comput. Syst., Berlin, Germany, 2013, pp. 867–873.
[12] R. Emery, T. Balch, Behavior-based control of a non-holonomic robot in pushing tasks, in: Proc. 2001 ICRA, IEEE Int. Conf. Robot. Autom. (Cat. No. 01CH37164), vol. 3, 2001, pp. 2381–2388.
[13] P. Glorennec, L. Jouffe, Fuzzy Q-learning, in: Proc. 6th Int. Fuzzy Syst. Conf., vol. 2, Barcelona, 1997, pp. 659–662.
[14] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, B. Maisonnier, Mechatronic design of NAO humanoid, in: 2009 IEEE Int. Conf. Robot. Autom., Kobe, Japan, 2009, pp. 769–774.
[15] I. Grondman, L. Busoniu, G.A.D. Lopes, R. Babuska, A survey of actor-critic reinforcement learning: standard and natural policy gradients, IEEE Trans. Syst. Man Cybern., Part C 42 (6) (2012) 1–17.
[16] G.C. How, T. Wu, M. Cutler, J.P. How, Rapid transfer of controllers between UAVs using learning-based adaptive control, in: Proc. – IEEE Int. Conf. Robot. Autom., Karlsruhe, Germany, 2013, pp. 5409–5416.
[17] K. Hwang, Y. Chen, C. Wu, Fusion of multiple behaviors using layered reinforcement learning, IEEE Trans. Syst. Man Cybern., Part A, Syst. Hum. 42 (4) (2012) 999–1004.
[18] A. Kabysh, V. Golovko, A. Lipnickas, Influence learning for multi-agent system based on reinforcement learning, Int. J. Comput. 11 (1) (2012) 39–44.
[19] M. Kaisers, K. Tuyls, Frequency adjusted multi-agent Q-learning, in: 9th Int. Conf. Auton. Agents Multiagent Syst., Toronto, Canada, 2010, pp. 309–315.
[20] H. Kimura, Reinforcement learning in multi-dimensional state-action space using random rectangular coarse coding and Gibbs sampling, in: IEEE/RSJ Int. Conf. Intell. Robot. Syst., 2007, pp. 88–95.
[21] M. Lauer, M. Riedmiller, An algorithm for distributed reinforcement learning in cooperative multi-agent systems, in: Int. Conf. Mach. Learn., Stanford, CA, USA, 2000, pp. 535–542.
[22] G.J. Laurent, L. Matignon, N.L. Fort-Piat, The world of independent learners is not Markovian, Int. J. Knowl. Based Intell. Eng. Syst. 15 (1) (2011) 55–64.
[23] D.L. Leottau, Decentralized reinforcement learning (source code), https://github.com/dleottau/Thesis-DRL, 2017.
[24] D.L. Leottau, C. Celemin, UCh-dribbling-videos, https://www.youtube.com/watch?v=HP8pRh4ic8w, 2015.
[25] D.L. Leottau, C. Celemin, J. Ruiz-del-Solar, Ball dribbling for humanoid biped robots: a reinforcement learning and fuzzy control approach, in: Reinaldo A.C. Bianchi, H. Levent Akin, Subramanian Ramamoorthy, K. Sugiura (Eds.), Rob. 2014 Robot World Cup XVIII, in: Lect. Notes Comput. Sci., vol. 8992, Springer Verlag, Berlin, 2015, pp. 549–561.
[26] D.L. Leottau, J. Ruiz-del-Solar, An accelerated approach to decentralized reinforcement learning of the ball-dribbling behavior, in: AAAI Work., vol. WS-15-09, Austin, Texas, USA, 2015, pp. 23–29.
[27] D.L. Leottau, J. Ruiz-del-Solar, P. MacAlpine, P. Stone, A study of layered learning strategies applied to individual behaviors in robot soccer, in: L. Almeida, J. Ji, G. Steinbauer, S. Luke (Eds.), RoboCup 2015: Robot Soccer World Cup XIX, in: Lect. Notes Artif. Intell., vol. 9513, Springer Verlag, Berlin, 2016, pp. 290–302.
[28] D.L. Leottau, A. Vatsyayan, J. Ruiz-del-Solar, R. Babuska, Decentralized reinforcement learning applied to mobile robots, in: S. Behnke, R. Sheh, S. Sariel, D.D. Lee (Eds.), Rob. 2016: Robot World Cup XX, in: Lect. Notes Artif. Intell., vol. 9776, Springer Verlag, Berlin, 2017.
[29] C.-K. Lin, A reinforcement learning adaptive fuzzy controller for robots, Fuzzy Sets Syst. 137 (3) (2003) 339–352.
[30] J. Martin, A reinforcement learning environment in Matlab (source code), https://jamh-web.appspot.com/download.htm.
[31] J. Martin, H.D. Lope, A distributed reinforcement learning architecture for multi-link robots, in: 4th Int. Conf. Informatics Control. Autom. Robot., ICINCO 2007, Angers, France, 2007, pp. 192–197.
[32] L. Matignon, G.J. Laurent, N. Le Fort-Piat, Design of semi-decentralized control laws for distributed-air-jet micromanipulators by reinforcement learning, in: 2009 IEEE/RSJ Int. Conf. Intell. Robot. Syst., 2009, pp. 3277–3283.
[33] L. Matignon, G.J. Laurent, N. Le Fort-Piat, Y.-A. Chapuis, Designing decentralized controllers for distributed-air-jet MEMS-based micromanipulators by reinforcement learning, J. Intell. Robot. Syst. 59 (2) (2010) 145–166.
[34] L. Panait, S. Luke, Cooperative multi-agent learning: the state of the art, Auton. Agents Multi-Agent Syst. 11 (3) (2005) 387–434.
[35] L. Panait, K. Sullivan, S. Luke, Lenience towards teammates helps in cooperative multiagent learning, in: Proc. Fifth Int. Jt. Conf. Auton. Agents Multi-Agent Syst., Hakodate, Japan, 2006.
[36] L. Panait, K. Tuyls, S. Luke, Theoretical advantages of lenient learners: an evolutionary game theoretic perspective, J. Mach. Learn. Res. 9 (2008) 423–457.
[37] S. Papierok, A. Noglik, J. Pauli, Application of reinforcement learning in a real environment using an RBF network, in: 1st Int. Work. Evol. Reinf. Learn. Auton. Robot Syst., ERLARS 2008, Patras, Greece, 2008, pp. 17–22.
[38] J. Pazis, M.G. Lagoudakis, Reinforcement learning in multidimensional continuous action spaces, in: IEEE Symp. Adapt. Dyn. Program. Reinf. Learn., Paris, France, 2011, pp. 97–104.
[39] E. Schuitema, Reinforcement Learning on Autonomous Humanoid Robots, Ph.D. Thesis, Delft University of Technology, 2012.
[40] S. Sen, M. Sekaran, J. Hale, Learning to coordinate without sharing information, in: Proc. Natl. Conf. Artif. Intell., American Association for Artificial Intelligence, Seattle, Washington, 1994, pp. 426–431.
[41] S.P. Singh, M.J. Kearns, Y. Mansour, Nash convergence of gradient dynamics in general-sum games, in: UAI '00 Proc. 16th Conf. Uncertain. Artif. Intell., Stanford, CA, 2000, pp. 541–548.
[42] P. Stone, M. Veloso, Multiagent systems: a survey from a machine learning perspective, Auton. Robots 8 (3) (2000) 1–57.
[43] R. Sutton, A. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.
[44] Y. Takahashi, M. Asada, Multi-layered learning system for real robot behavior acquisition, in: V. Kordic, A. Lazinica, M. Merdan (Eds.), Cut. Edge Robot., Intech, Germany, 2005, pp. 357–375.
[45] I. Tanev, T. Ray, A. Buller, Automated evolutionary design, robustness, and adaptation of sidewinding locomotion of a simulated snake-like robot, IEEE Trans. Robot. 21 (4) (2005) 632–645.
[46] M.E. Taylor, G. Kuhlmann, P. Stone, Autonomous transfer for reinforcement learning, in: Auton. Agents Multi-Agent Syst. Conf., Estoril, Portugal, 2008, pp. 283–290.
[47] E. Theodorou, J. Buchli, S. Schaal, Reinforcement learning of motor skills in high dimensions: a path integral approach, in: 2010 IEEE Int. Conf. Robot. Autom., 2010, pp. 2397–2403.
[48] S. Troost, E. Schuitema, P. Jonker, Using cooperative multi-agent Q-learning to achieve action space decomposition within single robots, in: 1st Int. Work. Evol. Reinf. Learn. Auton. Robot Syst., ERLARS 2008, Patras, Greece, 2008, pp. 23–32.
[49] K. Tuyls, P.J.T. Hoen, B. Vanschoenwinkel, An evolutionary dynamical analysis of multi-agent learning in iterated games, Auton. Agents Multi-Agent Syst. 12 (1) (2005) 115–153.

[50] A. Vatsyayan, Video: centralized and decentralized reinforcement learning of the ball-pushing behavior, https://youtu.be/pajMkrf7ldY, 2016.
[51] M. Veloso, P. Stone, Video: RoboCup robot soccer history 1997–2011, in: 2012 IEEE/RSJ Int. Conf. Intell. Robot. Syst., Vilamoura-Algarve, Portugal, 2012, pp. 5452–5453.
[52] N. Vlassis, A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence, vol. 1, Morgan and Claypool Publishers, 2007.
[53] C.J.C.H. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (3–4) (1992) 279–292.
[54] S. Whiteson, N. Kohl, R. Miikkulainen, P. Stone, Evolving keepaway soccer players through task decomposition, in: E. Cantú-Paz, J. Foster, K. Deb, L. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. Potter, A. Schultz, K. Dowsland, N. Jonoska, J. Miller (Eds.), Genet. Evol. Comput., GECCO 2003 SE-41, in: Lecture Notes in Computer Science, vol. 2723, Springer, Heidelberg, Berlin, 2003, pp. 356–368.
[55] J.M. Yanez, P. Cano, M. Mattamala, P. Saavedra, D.L. Leottau, C. Celemin, Y. Tsutsumi, P. Miranda, J. Ruiz-del-Solar, UChile Robotics Team, Team description for RoboCup 2014, in: Rob. 2014 Robot Soccer World Cup XVIII Preproceedings, Joao Pessoa, Brazil, 2014.