UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FíSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

MULTI-AGENT BASED DECENTRALIZED REINFORCEMENT LEARNING OF INDIVIDUAL BEHAVIORS

TESIS PARA OPTAR AL GRADO DE
DOCTOR EN INGENIERÍA ELÉCTRICA

DAVID LEONARDO LEOTTAU FORERO

PROFESOR GUÍA:
JAVIER RUIZ DEL SOLAR SAN MARTÍN

MIEMBROS DE LA COMISIÓN:
JORGE SILVA SÁNCHEZ
EDUARDO MORALES MANZANARES
REINALDO DA COSTA BIANCHI

SANTIAGO DE CHILE
2018

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE DOCTOR EN INGENIERÍA ELÉCTRICA
POR: DAVID LEONARDO LEOTTAU FORERO
FECHA: 2018
PROF. GUÍA: JAVIER RUIZ DEL SOLAR SAN MARTÍN

MULTI-AGENT BASED DECENTRALIZED REINFORCEMENT LEARNING OF INDIVIDUAL BEHAVIORS

El paradigma del aprendizaje reforzado (RL [1]) está siendo recurrentemente utilizado para aprender tareas complejas en contextos aplicativos como la robótica. No obstante, muchas aplicaciones en el mundo real manejan un espacio de acciones multi-dimensional, en donde las acciones individuales trabajan juntas para que el agente ejecute un comportamiento deseado. En tales aplicaciones, el RL presenta de una explosión en la complejidad computacional, que ocurre cuando se usan esquemas con RL Centralizado (CRL). Esto genera problemas como el consumo excesivo de memoria o de tiempo de entrenamiento, sin embargo, el uso del RL Descentralizado (DRL) ayuda a solucionar tales problemas. En esta tesis, se usará el termino DRL para referirse a aquellos métodos descentralizados usados para el aprendizaje de tareas ejecutadas por una entidad individual, por ejemplo, un robot.

En esta tesis, se propone una metodología para modelar el DRL de comportamientos individuales en problemas con espacios de acciones multi-dimensionales. Cada sub-problema (ej., las acciones de un efector o actuador) es aprendido por agentes independientes que trabajan en paralelo.

Dado que la mayoría de los estudios reportados sobre Sistemas Multi-Agente (MAS) no validan los métodos propuestos en problemas estocásticos-multiestado y del mundo real, uno de los objetivos de la presente tesis es demostrar empíricamente que los beneficios de los MAS son extensibles a problemas complejos como son las plataformas robóticas, si estos son modelados e implementados con sistemas DRL. Para ello, varios esquemas de DRL basados en algoritmos multi-agente y métodos de transferencia de conocimiento son presentados, validados y analizados, a través de un extenso estudio experimental, en donde diferentes problemas son modelados e implementados, siguiendo la metodología propuesta. Los resultados de la validación empírica muestran que las implementaciones con DRL mejoran el desempeño de su análogo CRL, usando además menos recursos computacionales. Además, aquellos esquemas de DRL implementados con mecanismos de coordinación, muestran mejores desempeños y/o tiempos de entrenamiento que los esquemas de DRL que no usan coordinación directa.

[1] Por sus siglas en inglés: Reinforcement Learning

Abstract

Reinforcement Learning (RL) is commonly used to learn complex behaviors. However, many real-world applications feature multi-dimensional action spaces, through which the individual actions work together to make the learning agent perform a desired task. In such applications, RL suffers from the combinatorial explosion of complexity, which occurs when a Centralized RL (CRL) scheme is used. 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 dissertation, it will be used the term DRL for decentralized approaches to the learning of a task which is performed by a single entity, e.g., a robot.

In this thesis, a Multi-Agent (MA) methodology is proposed for modeling the DRL of individual behaviors 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, one of the goals is to show empirically that the benefits of MAS are also applicable to complex problems like robotic platforms, by using DRL systems. In order to address this, several MA and Knowledge Transfer (KT) based DRL schemes are introduced, validated and analyzed through an extensive experimental study, in which different problems are modeled and implemented following the proposed methodology. Results from the empirical validation provides evidence that DRL implementations outperform their CRL counterparts, while using less computational resources. Also, that DRL schemes implemented with coordination mechanisms show better performance and/or faster learning times than DRL schemes with non direct coordination.

Agradecimientos

Gracias infinitas a todas mis madres: Susy, Mami Ceci, Rois y Stella. Su apoyo siempre me ha dado la fortaleza, confianza y entereza para emprender cada idea que se me cruza por la cabeza. A mi familia... con su afecto todo es más fácil.

Mis agradecimientos al Profesor Javier Ruiz del Solar que supo guiarme dándome la medida justa de libertad, apoyo, crítica y dirección. A Carlos Celemin, Patrick MacAlpine, Kenzo Lobos, Francisco Jaramillo, Pablo Guerrero, Peter Stone y Robert Babuska por haberme colaborado con el desarrollo de mi trabajo. A los integrantes del UChile Robotics Team, y a los compañeros del AMTC.

Gracias a Rous, Vane, Ros, Pancho y a todo el grupo por ser soporte cuando la familia está lejos. Gracias a todos esos amigos que han estado ahí en la música, el fútbol y las rutas$_h$. Ellos saben quiénes son. Sin deportes, risas y rock & roll todo esto hubiera sido aún más difícil.

Gracias al Advanced Mining Technology Center (AMTC) y a la Comisión Nacional de Investigación Científica y Tecnológica (CONICYT), por su auspicio por medio del Proyecto FONDECYT 1161500 y la beca CONICYT-PCHA/Doctorado Nacional/2013-63130183. Finalmente gracias Chile por acogerme todos estos años.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Definition of the Problem

Reinforcement Learning is commonly used in robotics to learn complex behaviors. Two of the main challenges to be solved for modeling RL systems acting in the real-world are: *(i)* the high dimensionality of the state and action spaces, and *(ii)* the large number of training trials required to learn most of complex tasks. 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 [9, 40], mobile robots [13, 34], aerial vehicles [3, 19], multi-legged robots [61], and snake robots [56]. In such applications, RL suffers from the combinatorial explosion of complexity, which occurs when a single-agent or Centralized RL (CRL) scheme is used [40]. It may turn infeasible to implement CRL systems in terms of computational resources or learning time due to the exponential increasing of dimensionality in both, the state space and the action space [40, 31]. Unlike, the use of *Decentralized Reinforcement Learning* (DRL) helps to alleviate this problem as it has been empirically evidenced [9, 32, 31].

In DRL, the learning problem is decomposed into several sub-problems, whose resources are managed separately, while working toward a common goal. However, the coordination problem must be solved, in order to extend and take advantage of some potential benefits of Multi-Agent Systems (MAS) [52]. In addition, most of the stochastic DRL systems implemented with independent learners also present two main drawbacks: non-stationary and non-Markovian issues.

1.2 Objectives

1.2.1 General Objective

The general objective of this thesis is to address the Multi-Agent Based Decentralized Reinforcement Learning of individual behaviors of those problems in which multi-dimensional action spaces are involved.

1.2.2 Specific Objectives

- To demonstrate empirically that an independent DRL system is able to achieve faster learning times and comparable performances regarding to its CRL counterpart.
- To generate a methodology for modeling and implementing DRL Systems to perform individual robot behaviors.
- To propose solutions to coordinate the individual learning agents and reduce the training time in DRL systems.

1.3 Hypothesis

When using DRL of individual behaviors, sub-problems are learned in parallel by individual agents working together. In the case of multidimensional action spaces, a sub-problem 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 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, taking advantage of parallel computation and other benefits of Multi-Agent Systems (MAS).

Most of the MAS reported studies do not validate the proposed approaches with multi-state, stochastic, and real-world problems, which limits the applicative field of MAS [7]. However, since a DRL system has several individual agents as part of the same entity, real-time communication and observation among those agents is not an issue unlike many of the MAS. Thus, DRL systems would be able to extend benefits and properties of MAS to challenging and real-world applications like complex robotic platforms.

1.4 Contributions

This thesis work presents several original contributions directly related of DRL of individual behaviors:

- A five stages methodology for modeling and implementing a DRL system is promoted and proposed, in which aspects such as what kind of problem is a candidate for being decentralized, which subproblems, actions, or states should or could be decomposed, what kind of reward functions and RL algorithms should be used, among other modeling issues, are addressed.

- The Cooperative Adaptive Learning Rate DRL (DRL-CA) algorithm is proposed, described and implemented. The main principle of DRL-CA is supported on a cooperative factor that adapts the learning rate on-line based on a simple estimation of the partial quality of the policy performed by the "weakest" agent.

- The Lenient Multi-Agent Reinforcement Learning algorithm has been implemented to multi-state, stochastic, and continuous state-action DRL problems, and it is described as DRL-Lenient.

- DRL-CA and DRL-Lenient algorithms, as well as DRL schemes with independent learners and no prior coordination (DRL-Ind), and their CRL counterparts are validated and evaluated through an extensive empirical study. For that purpose, the proposed five stages methodology for modeling and implementing DRL systems is applied and described in four different problems; two of them are well-known problems: an extended version of 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: the Ball-Dribbling in soccer performed with an humanoid biped robot, and the Ball-Pushing behavior performed with a differential drive robot. A deep analysis about their strengths and weaknesses are drawn according to each validation problem and their characteristics. To the best of our knowledge, this work is the first one that applies a decentralized modeling to the learning of individual behaviors on mobile robot platforms, and compares it with their centralized RL scheme counterparts.

- The Control Sharing (CoSh) knowledge transfer approach has been extended from the single-agent case to the DRL proposed architecture. Hence, the DRL+CoSh scheme is described and implemented.

- The Nearby Action Sharing method (DRL+NeASh) for continuous action spaces is proposed, described and implemented. DRL+NeASh is a variant of DRL+CoSh which is able to include a measure of uncertainty to the transferred action for noisy sources of knowledge.

- DRL+CoSh and DRL+NeASh algorithms are validated and analyzed through an extensive empirical study carried out in a 3D realistic simulator and demonstrated with physical robots. To the best of our knowledge, this work is the the first applying an effective strategy for transferring knowledge and coordinating-accelerating DRL systems.

- All the source codes are shared online, including the algorithms: DRL-Ind, DRL-CA, DRL-Lenient, DRL+CoSh, and DRL+NeASh; as well as the 3DMC, ball-pushing, ball-dribbling, and SCARA-RTG environments. A custom hill-climbing algorithm for optimizing RL parameters is also available.

- I expect that the proposed decentralized extension of the 3DMC can be used in future work as a test-bed for DRL and multi-state MAL problems.

1.4.1 Bibliographic and technical production

Journal papers

1. Decentralized Reinforcement Learning of Robot Behaviors [31].
2. Accelerating Decentralized Reinforcement Learning of Complex Individual Behaviors [30].

Proceedings papers

1. Toward Real-Time Decentralized Reinforcement Learning using Finite Support Basis Functions [38].
2. Decentralized Reinforcement Learning Applied to Mobile Robots [32].
3. A Study of Layered Learning Strategies Applied to Individual Behaviors in Robot Soccer [35].
4. An Accelerated Approach to Decentralized Reinforcement Learning of the Ball-Dribbling Behavior [34].
5. Ball Dribbling for Humanoid Biped Robots: A Reinforcement Learning and Fuzzy Control Approach [29].
6. Integration of the ROS Framework in Soccer Robotics: the NAO Case [33].

1.5 Structure of this Document

This thesis document is organized as follows: the present Chapter introduces the reader to the topic being addressed and to the dissertation work itself. Chapter 2 provides background information on single-agent RL, MARL, and independent learning, useful for understanding further chapters. Chapter 3 provides an introduction to DRL, some DRL algorithms are described, and an overview on related work is provided. Chapter 4 proposes a methodology to model and implement DRL systems. In Chapter 5, the DRL methodology introduced in Chapter 4, as well as the DRL algorithms described in Chapter 3 are validated through an experimental study on four different problems. Chapter 6 introduces two KT based DRL schemes, validating and discussing them by implementing two real-world problems. Finally, conclusions based on the methodologies developed and results obtained are drawn, as well as some ideas for future work. Additionally, Appendix A describes the custom hill-climbing algorithm used for optimizing RL parameters.

Chapter 2

Background

This chapter provides some concepts and background information useful for understanding further chapters. Section 2.1 presents information on single-agent RL based on Sutton and Barto [53], and Busnoniu, Babuska, De Schutter, and Ernst's [8] books. Section 2.2 provides background on multi-agent RL, based on Busnoniu et al. [7], and Vlassis's [66] articles. And, Section 2.3 gives a brief overview on independent learning, based on Laurent, Matignon, and Fort-Piat's [26] article.

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 document, 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 [7].

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.

The agent's goal is to maximize, at each time-step k, the expected discounted return:

$$R_k = \sum_{j=0}^{\infty} \gamma^j r_{k+j+1} \tag{2.1}$$

in this infinite-horizon reward case, $\gamma \in (0,1)$ is the discount factor. The task of the agent is, therefore, to maximize its long-term performance, while only receiving feedback about its immediate, one-step performance. One way it can achieve this is by computing an optimal *action-value function* (Q-function), in which $Q^\pi : S \times A \to \mathbb{R}$, is the expected return of a state-action pair given the policy $\pi : Q^\pi(s,a) = E\{\sum_{j=0}^{\infty} \gamma^j r_{k+j+1} | s_k = s, a_k = a, \pi\}$.

The agent can maximize its return and achieve the learning goal by first computing the optimal Q-function, which is defined as $Q^*(s,a) = \max_\pi Q^\pi(s,a)$, and then choosing actions by the greedy policy $\pi^*(s) = \text{argmax}_a Q^*(s,a)$, which is optimal (i.e. maximizes the expected return) when applied to Q^*. It satisfies the Bellman optimality recursion:

$$Q^*(s,a) = R(s,a,s') + \gamma \max_{a' \in A} Q^* \langle f(s,a,s'), a' \rangle \ \forall \ s, a. \tag{2.2}$$

There are a wide variety of single-agent RL algorithms reported in the literature. Two classes of RL approaches can be pointed: *value iteration*, an offline, model-based algorithm that learns the optimal value function when the transition model is available; and *policy iteration* algorithms in which an optimal policy is directly built by interacting with the environment [53].

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. On-line and model-free algorithms are considered, as they are convenient for practical implementations.

Q-Learning [68] 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:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \big[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a) \big] \tag{2.3}$$

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. This means that the agent must sometimes explore, i.e. perform other actions than those dictated by the current greedy policy.

A common exploration policy is the so-called ε-greedy policy by which in state s the agent selects a random action with probability ε, and action $a = \text{argmax}_{a'} Q(s,a')$ with probability $1-\varepsilon$, where $\varepsilon \in [0,1]$. One drawback of ε-greedy is that when it explores, it chooses equally among all actions. A solution is to vary the action probabilities as a graded function of estimated value. The greedy action is still given the highest selection probability, but all the

others are ranked and weighted according to their value estimates. These are called *Softmax* action selection rules. The most common softmax method uses a Gibbs, or Boltzmann, distribution. It chooses action a on the k_{th} play with probability:

$$p(a|s) = \frac{\exp(Q(s,a)/\tau)}{\sum_{a'} \exp(Q(s,a')/\tau)} \tag{2.4}$$

where $\tau > 0$ is a parameter called the temperature. High temperatures cause the actions to be all (nearly) equiprobable. Low temperatures cause a greater difference in selection probability for actions that differ in their value estimates. In the limit as $\tau \leftarrow 0$, softmax action selection becomes the same as greedy action selection. When $\tau \leftarrow \infty$, action selection is purely random. τ can be decreased with time.

2.2 Multi-Agent Reinforcement 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 $\mathcal{A} = A^1 \times \cdots \times A^M$; $T : S \times \mathcal{A} \times S \rightarrow [0, 1]$ the state transition probability function, such that, $\forall s \in S, \forall a \in \mathcal{A}, \sum_{s' \in S} T(s, a, s') = 1$; and $R^m : S \times \mathcal{A} \times S \rightarrow \mathbb{R}, m = 1, \cdots, M$ the reward functions of the agents [7, 26].

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 $\pi^m : S \times A^m \rightarrow [0, 1]$ form together the joint policy π. The Q-function of each agent depends on the joint action and is conditioned on the joint policy, $Q_m^\pi : S \times \mathcal{A} \rightarrow \mathbb{R}$.

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 [66]. 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 [9].

2.3 Independent Learners

Claus and Boutilier [10] 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 [26].

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 [26].

For ILs, which is the focus of the present document, the individual policies change as the learning progresses. Therefore, the environment is non-stationary and non-Markovian. Laurent, Matignon and Fort-Piat [26] 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 [26].

Chapter 3

Decentralized Reinforcement Learning

This chapter introduces the DRL and some practical algorithms, which is the core of this dissertation. In this book, the term DRL will be used for decentralized approaches to the learning of a task which is performed by a single entity, e.g., a robot. The remainder of this chapter is organized as follows: Section 3.1 provides an introduction to DRL, its potential advantages are pointed in Section 3.2, and some challenges are specified in Section 3.3. Three DRL algorithms are described in Section 3.4. Finally, some related work is discussed in Section 3.5.

This chapter is fully based on our paper [31]: *Decentralized Reinforcement Learning of Robot Behaviors*, which is in press by the *Artificial Intelligence Journal*.

3.1 An Introduction to DRL

From a Distributed Artificial Intelligence perspective, the Distributed Problem Solving subfield focuses on the information management aspects of systems with several branches working together toward a common goal; for example, problems such as task decomposition and solution synthesis can often be decomposed into several not entirely independent sub-problems that can be solved on different processors. On the other hand, MAS deal with behavior management in collections of several independent entities, or agents [52].

DRL is concerned with MAS and Distributed Problem Solving. 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. In the case of multidimensional action spaces, a subproblem corresponds to controlling one particular variable. If each variable is controlled individually, a distributed control scheme can be applied. Through coordination of the individual learning agents, it is possible to use decentralized methods [9], taking advantage of parallel computation and other benefits of Multi-Agent Systems (MAS) [52, 7].

For instance, consider a quadcopter learning to perform a maneuver: each rotor can be

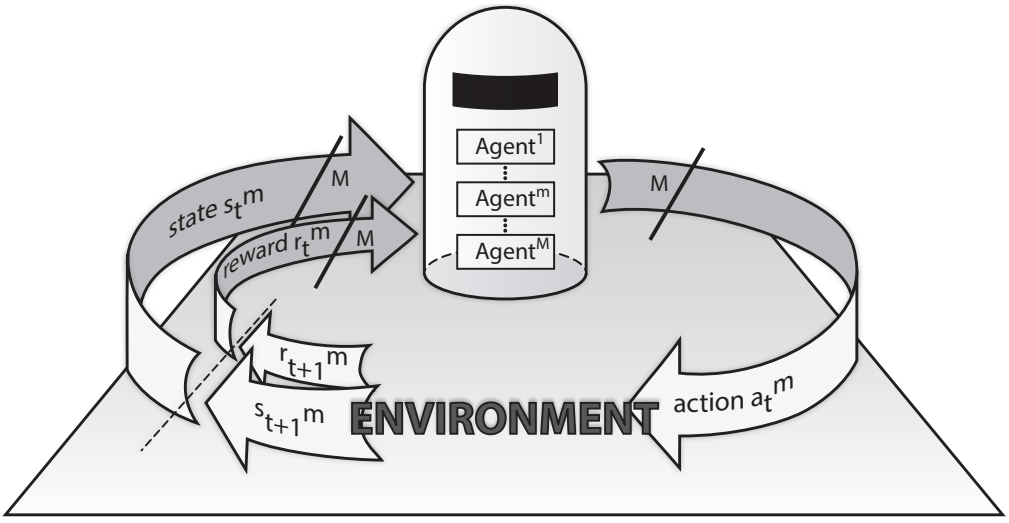

Figure 3.1: The basic DRL architecture.

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 [9], where it was used to differentiate a decentralized system from a MAL system composed of individual agents [44]. The basic DRL architecture is shown in Figure 3.1 where M individual agents are interacting within an environment. According to Tuyls, Hoen, and Vanschoenwinkel [63], 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 dissertation that assumption is empirically validated 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.

3.2 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 [40]. 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 [49], this

is one of the main benefits of using DRL over CRL schemes, expressed by the following ratio:

$$\frac{\prod_{m=1}^{M} |N^m|}{\sum_{m=1}^{M} |N^m|}, \tag{3.1}$$

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 (3.1) [49].
- 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 [69], behavior fusion [20], and layered learning [55]. However, in this work is proposed 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.

3.3 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 2.3, 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 [7]. 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 [26]. In Section 3.4, 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.

3.4 DRL Algorithms

Some relevant MAL algorithms from state-of-the-art have been implemented and tested. These algorithms accomplish the three basic requirements of our interest: (i) no prior co-

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

(a) *Distributed Q-Learning* [25]: 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* [43]: it was 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) [51] and the *Win or Learn Fast* (WoLF) principle [5]: 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) [2]: 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 SARSA* without sharing information (e.g., the one reported by Sen, Sekaran, and Hale [50]): it mostly showed asymptotic convergence.

(f) *Lenient Multi-Agent Reinforcement Learning* [45]: it showed asymptotic convergence when applied to multi-state DRL problems.

From the above, in the present study, the following three algorithms are chosen: (i) ***Independent DRL*** (DRL-Independent), similar to (e) but implemented with decayed exploration and replacing traces; (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, the author's 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 5.1.

3.4.1 Independent DRL (DRL-Ind)

DRL-Ind aims to apply single-agent RL methods to the MARL task, and does not consider any kind of cooperation or coordination among agents; there is neither adaptation to the other agents nor estimated models of their policies, nor special action-selection mechanisms (e.g., communication among agents, prior knowledge). The computational complexity of this DRL scheme is the same as that for a single-agent RL (e.g., a Q-Learner).

Although the non-stationarity of the MARL problem invalidates most of the single-agent RL theoretical guarantees, this approach has been implemented in several multi-robot systems. An empirical study about DRL-Ind effectiveness can be found in [31].

Algorithm 3.1 depicts an episodic multi-agent SARSA algorithm [53] for continuous states with RBF approximation [47], built following the DRL-Ind scheme. There, a learning system with an M-dimensional action space is modeled with M single SARSA learners acting in

parallel. Every IL has individual Q-functions, action spaces, action selection mechanisms, and state vectors. Taking advantage of parallel computation of MAS, it is possible to update every Q-table by using M independent threads (Lines 3.1.18 to 3.1.35). A decayed and synchronized exploration rate is proposed, in order to avoid too much concurrent exploration and reduce the non-Markovian effects as was suggested in Section 3.3. In this way, each agent should find the best response to the behavior of the others. Thus, an ε-greedy action selection mechanism is implemented, which is exponentially decayed by using the d*ec* factor as seen in Line 3.1.29; e*piso*de is the current episode index and *maxEpiso*de*s* is the total number of trained episodes per run.

Synchronizing the exploration-exploitation mechanism among all the agents is a variant that Algorithm 3.1 offers. It is possible just by declaring a unique random number for all the agents as in Line 3.1.14, instead of an individual random scalar per agent as in Line 3.1.16. Also note that the RL parameters could be defined separately per agent (e.g., α^m, γ^m, ε^m), which is one of the DRL properties pointed out in Section 3. In Algorithm 3.1, those parameters are unified just for the sake of simplicity.

3.4.2 Lenient DRL (DRL-Lenient)

Originally proposed by Panait *et al.* [45], 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 [46] 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

$$\textbf{if } (U_{a^*} \leq r) \textbf{ or } urnd < 10^{-2} + \kappa^{-\beta\tau_{a^*}} \textbf{ then } U_{a^*} \leftarrow \alpha U_{a^*} + (1-\alpha)r, \tag{3.2}$$

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. (3.2) 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.* [62] and Schuitema [49], a multi-state implementation of Lenient Q-learning can be accomplished by combining the Q-Learning update rule (i.e. Eq. (2.3)) with the optimistic assumption proposed by Lauer and Riedmiller [25]. 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. (2.3)):

$$Q(s_t, a_t) \leftarrow \begin{cases} Q(s_t, a_t) + \alpha\delta, & \text{if } \delta > 0 \text{ or } urnd > \ell(s_t, a_t), \\ Q(s_t, a_t), & \text{otherwise,} \end{cases} \tag{3.3}$$

Algorithm 3.1 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:
    if Synchronized exploration **then**
      $urnd \leftarrow$ a uniform random variable $\in [0, 1]$
    else
      $[urnd^1, \cdots, urnd^M] \leftarrow$ a uniform random vector $\in [0, 1]$
    end if
    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 $s' = s'^m$
      if $urnd^m > \varepsilon$ **then**
        for all action i $\in A^m(s')$ **do**
          $Q_{\mathrm{i}} \leftarrow \sum_{j=1}^{\Phi^m} \theta^m_{\mathrm{i}}(j)\phi^m_{s'}(j)$
        end for
        $a' \leftarrow \mathrm{argmax_i} Q_{\mathrm{i}}$
      else
        $a' \leftarrow$ a random action $\in A^m(s')$
      end if
      $\varepsilon = \varepsilon_0 \exp(-\mathrm{dec} \cdot episode/maxEpisodes)$
      $Qas = \sum_{j=1}^{\Phi^m} \theta^m_a(j)\phi^m_s(j)$
      $Qas' = \sum_{j=1}^{\Phi^m} \theta^m_{a'}(j)\phi^m_{s'}(j)$
      $\delta \leftarrow r^m + \gamma Qas' - Qas$
      $\theta^m_a \leftarrow \theta^m_a + \alpha\delta\phi^m_s$
      $s^m \leftarrow s', a^m \leftarrow a'$
    end for
  until Terminal condition
end procedure

with $\delta \leftarrow r + \gamma Q' - Q$, and the state-action pair dependent lenience $\ell(s_t, a_t)$ 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 this study, lenient learning is implemented by adapting the update rule (3.3) to multi-state, stochastic, continuous state-action DRL problems, as reported by Troost *et al.* [62] and Schuitema [49] . The DRL-Lenient algorithm presented in Algorithm 3.2, which is implemented by replacing traces, incorporates a tabular MA-SARSA(λ) method, and uses softmax action selection from Sutton and Barto [53].

In Algorithm 3.2, 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 $\tau(s_t, a_\mathrm{i})$ 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 $\mathrm{e}(s,a)$, and the action-value functions. Following a linear gradient-descent strategy with RBF-features, similar to that presented in Algorithm 3.1, function approximations can be expressed as:

$$\mathrm{e}_a \leftarrow \mathrm{e}_a + \phi_s, \tag{3.4a}$$
$$\tau(s,a) = \sum_{j=1}^{\Phi} \tau_a(j)\phi_s(j), \tag{3.4b}$$
$$\tau_a \leftarrow \tau_a - (1-\beta)\tau(s,a)\phi_s, \tag{3.4c}$$
$$\delta \leftarrow r + \gamma \sum_{j=1}^{\Phi} \theta_{a'}(j)\phi_{s'}(j) - \sum_{j=1}^{\Phi} \theta_a(j)\phi_s(j), \tag{3.4d}$$
$$\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{\mathrm{e}}, \tag{3.4e}$$
$$\vec{\mathrm{e}} \leftarrow \gamma\lambda\vec{\mathrm{e}}, \tag{3.4f}$$

where Φ is the size of the feature vector ϕ_s. Equations (3.4a), (3.4c), (3.4d), (3.4e) and (3.4f) would approximate lines 19, 29, 28, 33 and 36, respectively. For practical implementations, τ_a must be set between $(0, 1)$.

Algorithm 3.2 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:
    $Q^m(s^m, a^m) = 0$, $\mathrm{e}^m(s^m, a^m) = 0$, 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 $s' = s'^m$
    $\mathrm{e}^m(s, a) \leftarrow 1$
    $min\tau \leftarrow \kappa(1 - \min\limits_{action}\limits^{N^m}_{\mathrm{i}=1}(\tau^m(s, a_\mathrm{i})))$
    $maxQv \leftarrow \max\limits_{action}\limits^{N^m}_{\mathrm{i}=1}(Q^m(s, a_\mathrm{i}))$
    for all action $\mathrm{i} \in A^m(s')$ **do**
      $Vqa_\mathrm{i} \leftarrow \exp(min\tau(Q^m(s, a_\mathrm{i}) - maxQv))$
    end for
    $Pa = [Pa_1, \cdots, Pa_{N^m}]$ ▷ Define probability distribution per-action at state s
    $Pa \leftarrow \dfrac{Vqa}{\sum_{\mathrm{i}=1}^{N^m} Vqa_\mathrm{i}}$
    Choose action $a' = a_{\mathrm{i}^*} \in \{1, \cdots, N^m\}$ ▷ 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**
        $Q^m(s, a) \leftarrow Q^m(s, a) + \alpha\delta\mathrm{e}^m(s, a)$
      end for
    end if
    $\mathrm{e}^m \leftarrow \gamma\lambda\mathrm{e}^m$
    $s^m \leftarrow s'; a^m \leftarrow a'$
  end for
until Terminal condition

3.4.3 Cooperative Adaptive DRL (DRL-CA)

This section introduces the DRL Cooperative Adaptive Learning Rate algorithm (DRL-CA), which mainly takes inspiration from the MARL approaches with a variable learning rate [5], and Frequency Adjusted Q-Learning (FAQL) [43]. The idea of a variable learning rate is used from the WoLF principle [5] and the IGA algorithm [51], in which agents learn quickly when losing, and cautiously when winning. The WoLF-IGA algorithm requires knowing the actual distribution of the 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 [5], 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 [43], in which the Q update rule

$$Q_{\mathrm{i}}(t+1) \leftarrow Q_{\mathrm{i}}(t) + \min\left(\frac{\beta}{Pa_{\mathrm{i}}}, 1\right) \alpha[r + \gamma \max_{j} Q_j(t) - Q_{\mathrm{i}}(t)] \tag{3.5}$$

is modified by the adjusted frequency parameter ($min(\beta/x_{\mathrm{i}}, 1)$). In the DRL-CA approach, such term is replaced by a cooperative adaptive factor ς defined as

$$\varsigma = 1 - \min_{agent} \prod_{m=1}^{M} Pa^{*,m} \tag{3.6}$$

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 [51]. In this way, DRL-CA shows a decreasing step size if a cooperative adaptive factor ς such as (3.6) is used. This decremental variation is referred 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, $\varsigma \to 1$ 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, $\varsigma \to 0$ and a decreasing learning rate, while better decisions are being made. Note that DRL-CAdec acts contrarily to the DRL-Lenient principle.

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

$$\varsigma = \min_{agent} \prod_{m=1}^{M} Pa^{*,m} \tag{3.7}$$

instead of (3.6). 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 5.1. The common principle behind both variants is the cooperative adaptation based on the current weakest learner's performance. I 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 3.3 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. (3.7)) is calculated in line 32, and the decremental cooperative adaptive factor (Eq. (3.6)) 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

$$Pa^{*',m} = Pa^{*,m} - \left[\frac{(N^m Pa^{*,m} - 1)}{(N^m(1 - N^m))} + \frac{1}{N^m}\right] \tag{3.8}$$

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 3.2 and 3.3 have been described with a softmax action selection

Algorithm 3.3 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: $\vec{\theta}^m = 0, \vec{\mathrm{e}}^m = 0, \tau = \tau_0$, 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 $s' = s'^m$

      $\mathrm{e}_a \leftarrow \mathrm{e}_a + \phi_s$

      $\delta \leftarrow r^m - \sum_{j=1}^{\Phi^m} \theta_a^m(j)\phi_s^m(j)$

      $Q_\mathrm{i} \leftarrow \sum_{j=1}^{\Phi^m} \theta_\mathrm{i}^m(j)\phi_{s'}^m(j)$ **for all** action $\mathrm{i} \in A^m(s')$

      $maxQv \leftarrow \max_{action}^{N^m} {}_{\mathrm{i}=1} Q_\mathrm{i}$

      $Vqa_\mathrm{i} \leftarrow \exp\left(\frac{(Q_\mathrm{i} - maxQv)}{(1+\tau)}\right)$ **for all** action $\mathrm{i} \in N^m$

      $Pa = [Pa_1, \cdots, Pa_{N^m}]$ ▷ probability distribution per-action at state s

      $Pa \leftarrow Vqa / \sum_{\mathrm{i}=1}^{N^m} Vqa_\mathrm{i}$

      Choose action $a' = a_{\mathrm{i}^*} \in \{1, \cdots, N^m\}$ ▷ at random using probability distribution $[Pa_1, \cdots, Pa_{N^m}]$

      $\delta \leftarrow \delta + \gamma Q_{\mathrm{i}^*}$

      $\vec{\theta}^m \leftarrow \vec{\theta}^m + \varsigma\alpha\delta\vec{\mathrm{e}}^{\,m}$

      $\vec{\mathrm{e}} \leftarrow \gamma\lambda\vec{\mathrm{e}}$

      $Pa^{*,m} \leftarrow Pa_{a'}$ ▷ Boltzmann probability of the selected action

      $s^m \leftarrow s'; a^m \leftarrow a'$

    end for

    $\tau = \tau_0 \exp(-\mathrm{dec} episode / maxEpisodes)$

    $\varsigma = \min_{agent}^{M} {}_{m=1}(Pa^{*,m})$ ▷ CAinc variation

    if CAdec variation **then**

      $\varsigma = 1 - \varsigma$

    end if

  until Terminal condition

end for

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.

3.5 Related Work

Busoniu *et al.* [9] 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 [40] 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 [62], 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 [49]. Moreover, the DRL of the soccer ball-dribbling behavior is accelerated by using knowledge transfer [34], where, each component of the humanoid 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 [35], 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 [13]. 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 [21], 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 [22] 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 [48] 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 [42], 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 [11], 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.

3.6 Summary

This chapter provided an introduction to DRL, its potential advantages and challenges. Three DRL algorithms were described: independent DRL scheme (DRL-Ind), which does not consider any kind of cooperation or coordination among the agents, applying single-agent RL methods to the multi-agent task; 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.

Chapter 4

Proposed Methodology for Modeling Decentralized Reinforcement Learning Systems

In this chapter is proposed a five stages methodology for modeling and implementing a DRL system. Aspects such as what kind of problem is a candidate for being decentralized, which subproblems, actions, or states should or could be decomposed, what kind of reward functions and RL algorithms should be used, among other modeling issues, are addressed. The 3DMC is used as a case study to draw an applicable example in this section.

This chapter is fully based on our paper [31]: *Decentralized Reinforcement Learning of Robot Behaviors*, which is in press by the *Artificial Intelligence Journal*.

4.1 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 5.2.

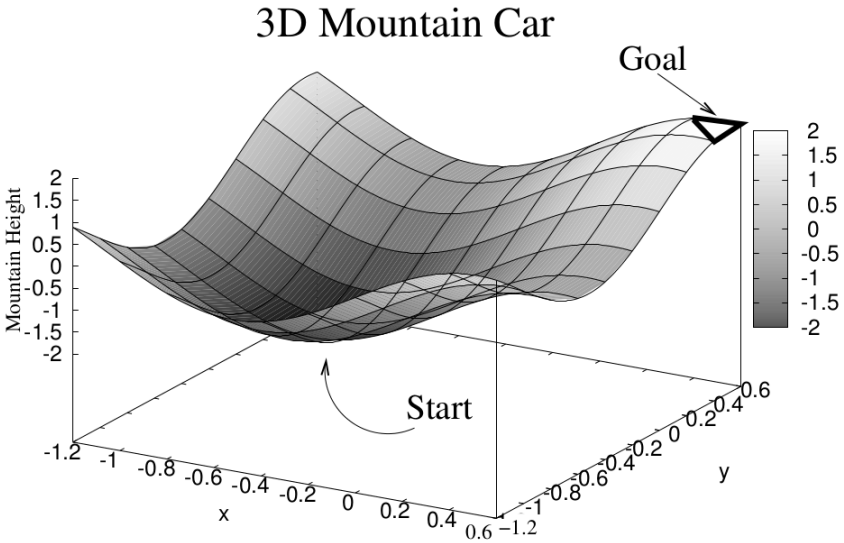

Figure 4.1: 3D mountain car surface. Figure adopted from Taylor and Stone [59].

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 [59], the mountain's curve is extended to a 3D surface as is shown in Figure 4.1. The state has four continuous state variables: $[x, \dot{x}, y, \dot{y}]$. The agent 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.

The 1D mountain car is an example of a non-decentralizable problem. This has a one-dimensional action space: $\{West, Neutral, East\}$, which is not splittable in parallel actions. It means, only one action can be executed per step. Unlike, the 3DMC 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 $Agent_y$.

4.2 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 Figure 4.1. Individual goals can be identified, $Agent_x$ should reach the east top, and $Agent_y$ should reach the north top.

4.3 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 [53]. 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.

4.4 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 4.1 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 4.2, as well as on each individual reward function designed as in Section 4.3. 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 4.3, 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, it has been 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 5.2.

4.5 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 [68] or R-MAX [6]; for a continuous state and discrete action space problem, a SARSA with function approximation [53] might be more useful; for a continuous state-action space problem, a Fuzzy Q-Learning [16] or an actor critic scheme [18] 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.

4.6 Summary

A methodology for modeling and implementing a DRL system has been presented in this chapter 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 4.4 rather than to determine individual goals in Stage 4.2. 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 Figure 4.2.

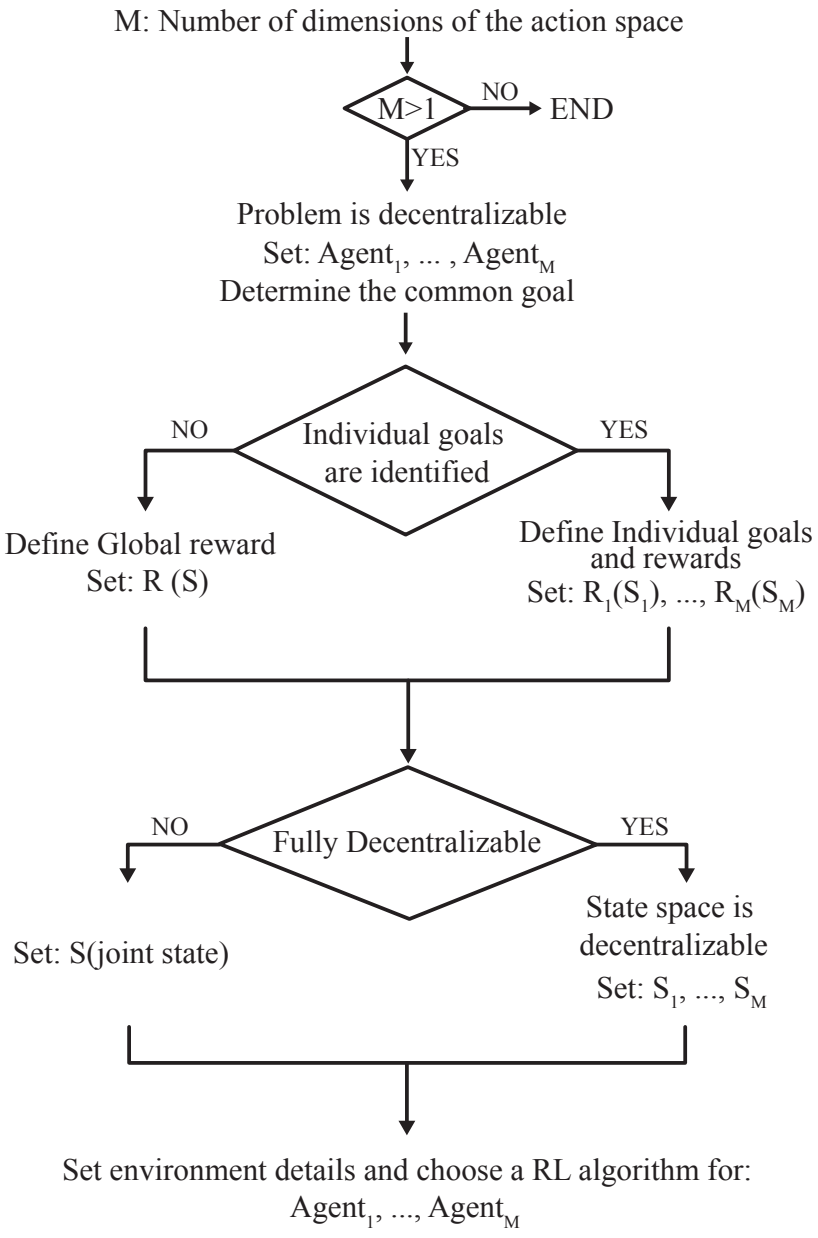

Figure 4.2: Proposed procedure for modeling a DRL problem.

Chapter 5

Experimental Validation of Decentralized Reinforcement Learning of Robot Behavios

This chapter examines some practical DRL algorithms applied to different multidimensional action space problems. For this, the three DRL algorithms described in Section 3.4 are considered: DRL-Independent, DRL-Lenient, and DRL-CA. These approaches are evaluated and analyzed through an empirical study using four different problems: 3D Mountain Car (Section 5.2), Ball-Pushing (Section 5.3), Ball-Dribbling (Section 5.4), and SCARA-RTG (Section 5.5). Finally, Section 5.6 discusses results.

This chapter is fully based on our paper [31]: *Decentralized Reinforcement Learning of Robot Behaviors*, which is in press by the *Artificial Intelligence Journal*.

5.1 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 [59]; the SCARA-RTG, a SCARA robot generating a real-time trajectory for navigating towards a 3D goal position [40]; the *Ball-Pushing* performed with a differential drive robot [32]; and the soccer *Ball-Dribbling* task [29]. 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 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 sub-sections, following the methodology described in Section 4. The experimental results and analysis are then discussed. All the acronyms of the implemented methods and problems are listed in Table 5.1. The following terminology is used: 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 [59]) and 9 actions 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 5.3). In the case of the Ball-Dribbling problem, DRL-Transfer is a DRL system accelerated by using the NASh transfer knowledge learning approach [34]; RL-FLC is an implementation reported by Leottau, Celemin, and Ruiz del Solar [29], 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 5.4).

5.2 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 [59], the mountain's curve is extended to a 3D surface as is shown in Figure 4.1.

Table 5.1: Experiment's acronyms and their optimized parameters

Acronym	**Optimized Parameters**
	3DMC
CRL-5a	$\alpha = 0.25, \lambda = 0.95, \varepsilon = 0.06$
CRL-9a	$\alpha = 0.20, \lambda = 0.95, \varepsilon = 0.06$
DRL-ObsF-Ind	$\alpha = 0.25, \lambda = 0.80, \varepsilon = 0.06$
DRL-ObsF-CAdec	$\alpha = 0.15, \lambda = 0.90, \varepsilon = 0.05$
DRL-ObsF-CAinc	$\alpha = 0.20, \lambda = 0.80, \varepsilon = 0.06$
DRL-ObsF-Lenient	$\alpha = 0.10, \lambda = 0.95, \varepsilon = 0.04, \kappa = 3.5, \beta = 0.8$
DRL-ObsL-Ind	$\alpha = 0.20, \lambda = 0.95, \varepsilon = 0.06$
DRL-ObsL-CAdec	$\alpha = 0.15, \lambda = 0.95, \varepsilon = 0.05$
DRL-ObsL-CAinc	$\alpha = 0.30, \lambda = 0.95, \varepsilon = 0.02$
DRL-ObsL-Lenient	$\alpha = 0.15, \lambda = 0.95, \varepsilon = 0.10, \kappa = 3, \beta = 0.75$
	Ball-Pushing
CRL	$\alpha = 0.50, \lambda = 0.90, \tau_0 = 2, \mathrm{dec} = 7$
DRL-Ind	$\alpha = 0.30, \lambda = 0.90, \tau_0 = 1, \mathrm{dec} = 10$
DRL-CAdec	$\alpha = 0.40, \lambda = 0.95, \tau_0 = 1, \mathrm{dec} = 10$
DRL-CAinc	$\alpha = 0.30, \lambda = 0.95, \tau_0 = 5, \mathrm{dec} = 13$
DRL-Lenient	$\alpha = 0.30, \lambda = 0.95, \ \kappa = 1, \quad \beta = 0.7$
DRL-Hybrid	$\alpha = 0.30, \lambda = 0.95,$ greedy
	Ball-Dribbling
CRL	$\alpha = 0.50, \lambda = 0.90, \varepsilon = 0.3, \mathrm{dec} = 10$
DRL-Ind	$\alpha = 0.50, \lambda = 0.90, \tau_0 = 70, \mathrm{dec} = 6$
DRL-CAdec	$\alpha = 0.10, \lambda = 0.90, \tau_0 = 20, \mathrm{dec} = 8$
DRL-CAinc	$\alpha = 0.30, \lambda = 0.90, \tau_0 = 70, \mathrm{dec} = 11$
DRL-Lenient	$\alpha = 0.10, \lambda = 0.90, \ \kappa = 1.5, \beta = 0.9$
DRL+Transfer	Final performance taken from Leottau *et al.* [34]
RL-FLC	Final performance taken from Leottau *et al.* [29]
eRL-FLC	Final performance taken from Leottau *et al.* [35]
	SCARA-RTG
DRL-Ind	$\alpha = 0.3, \varepsilon = 0.01$
DRL-CAdec	$\alpha = 0.3, \varepsilon = 0.01$
DRL-CAinc	$\alpha = 0.3, \varepsilon = 0.01$
DRL-Lenient	$\alpha = 0.3, \varepsilon = 0.01, \kappa = 2.0, \beta = 0.8$

5.2.1 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. An 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 $5,000$ time steps.

CRL-9a: The original centralized modeling (CRL-5a) [59] 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, the problem is extended, 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.

5.2.2 Proposed Decentralized Modelings

The methodology proposed in Section 4 is followed, resuming and extending the 3DMC DRL modeling:

Stage 4.1 Determining if the problem is decentralizable: CRL-5a is an example of a non-decentralizable modeling, however, due to the bidimentional action space of the 3DMC problem $(\dot{x}, \dot{y})$, the CRL-9a modeling is decentralizable 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 4.2 and 4.3 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 4.4 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. So, DRL-ObsL's implementations correspond to fully decentralizable modelings, while DRL-ObsF's are not, because DRL-ObsF's use a joint state vector.

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 5.1 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 (Fully decentralized): $agent_x = [x, \dot{x}]$, with $[9, 6]$ RBF cores; $agent_y = [y, \dot{y}]$, with $[9, 6]$ RBF cores.

Stage 4.5 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.

5.2.3 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.

5.2.4 RL Algorithm and Optimized Parameters

SARSA(λ) with Radial Basis Function (RBF) approximation with ε-greedy exploration [53] 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 5.1 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 $\frac{1}{2}\frac{|feature_{max} - feature_{min}|}{nCores}$. For all the experiments $\gamma = 0.99$.

5.2.5 Results and Analysis

Figure 5.1 (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 5.3 shows averaged final performances. Results presented here for CRL-5a converge considerably faster than the results presented by Taylor and Stone [59], which could be due to parameter optimization, and because an RBF approach was used 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 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 $2 \times 9 \times 6 \times 9 \times 6 \times 3$ memory cells, versus $9 \times 6 \times 9 \times 6 \times 9$ of its CRL-9a counterpart; and DRL-ObsF-Ind consumes $1/3$ less memory. Moreover, the elapsed time was measured of both learning process along the 25 performed runs, and founds that the DRL was 1.56 times faster than CRL. It was also measured only the action-selection+Q-function-update averaged elapsed time, obtaining that DRL process is 1.43 times faster than the CRL scheme. Those measures are compiled in Table 5.2 These times are referential; experiments with an Intel(R)Core(TM)i7-4774CPU@3.40Ghz with 4GB in RAM were performed. Note than even for this simple problem with only two agents, there are considerable memory consumption and processing time savings.

Approach	**Memory cells**	**Elapsed time**	**Action-Selection time**
DRL-ObsF-Ind	17,496	0.62 hrs	306.81 sec
CRL-9a	26,244	0.97 hrs	439.59 sec

Table 5.2: DRL vs. CRL computational consumption for 3DMC

Figure 5.1 (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-5a and CRL-9a presented in Figure 5.1 (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 Figure 5.1, all the DRL systems accelerate the asymptotic convergence considerably and outperform the CRL ones. Note also that all the DRL systems show similar learning times, while in Table 5.3, DRL-ObsF-CAdec shows the best performance, overcoming the -200 performance threshold with DRL-ObsF-Lenient.

Figure 5.1 (bottom) shows a performance comparison between schemes implemented considering limited observability (ObsL) of the joint space state, these schemes are: CRL-5a;

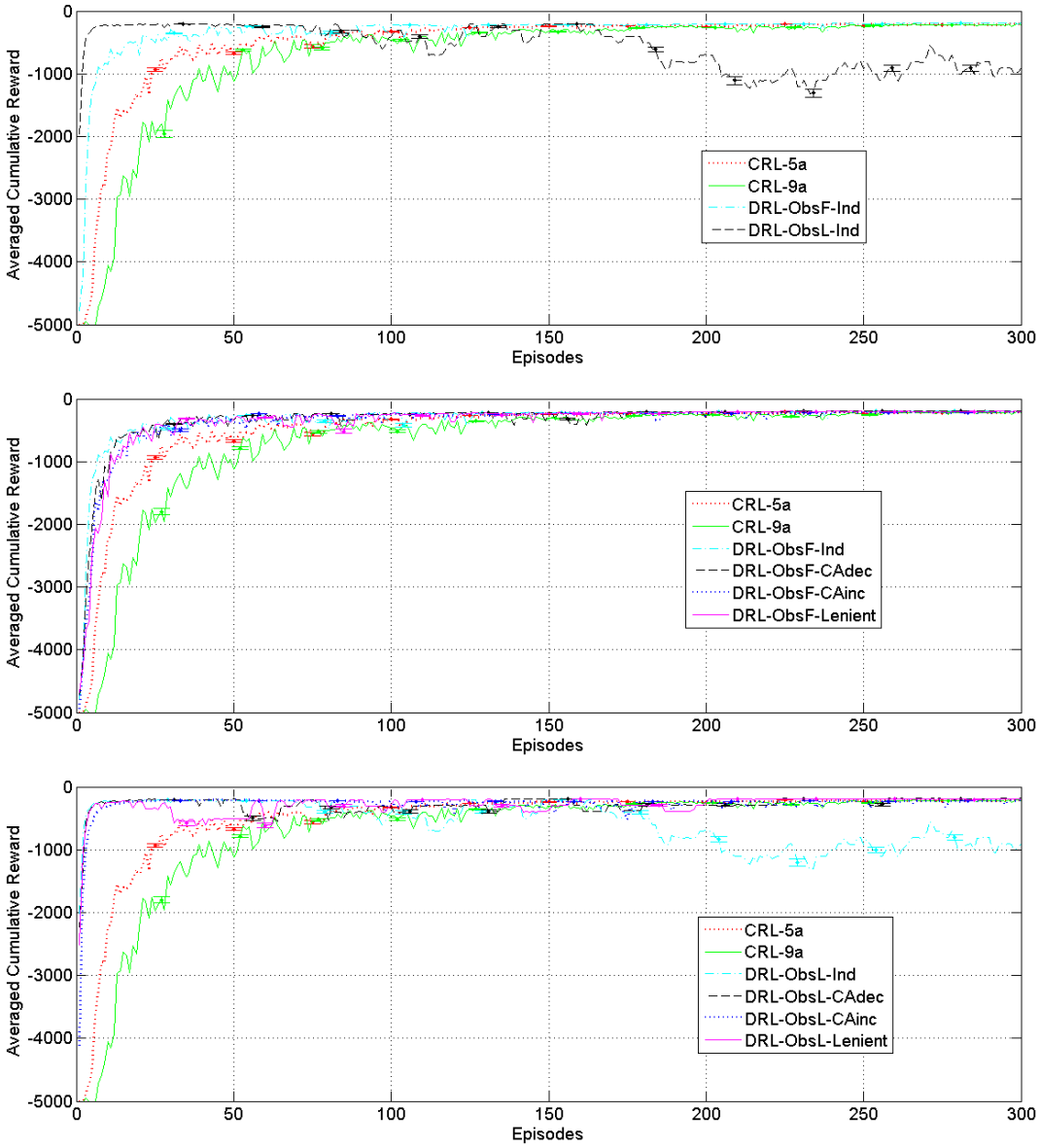

Figure 5.1: 3DMC learning evolution plots: centralized vs. decentralized approaches (top); centralized vs. decentralized approaches with full observability of the joint state space (middle); centralized vs. decentralized approaches with limited observability (bottom).

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 schemes. 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 5.3 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.

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

Table 5.3: 3DMC performances (these improve toward zero)

5.3 Ball-pushing

The Ball-Pushing behavior is considered, a basic robot soccer skill similar to that studied by Takahashi and Asada [55] and Emery and Balch [14]. A differential robot player attempts

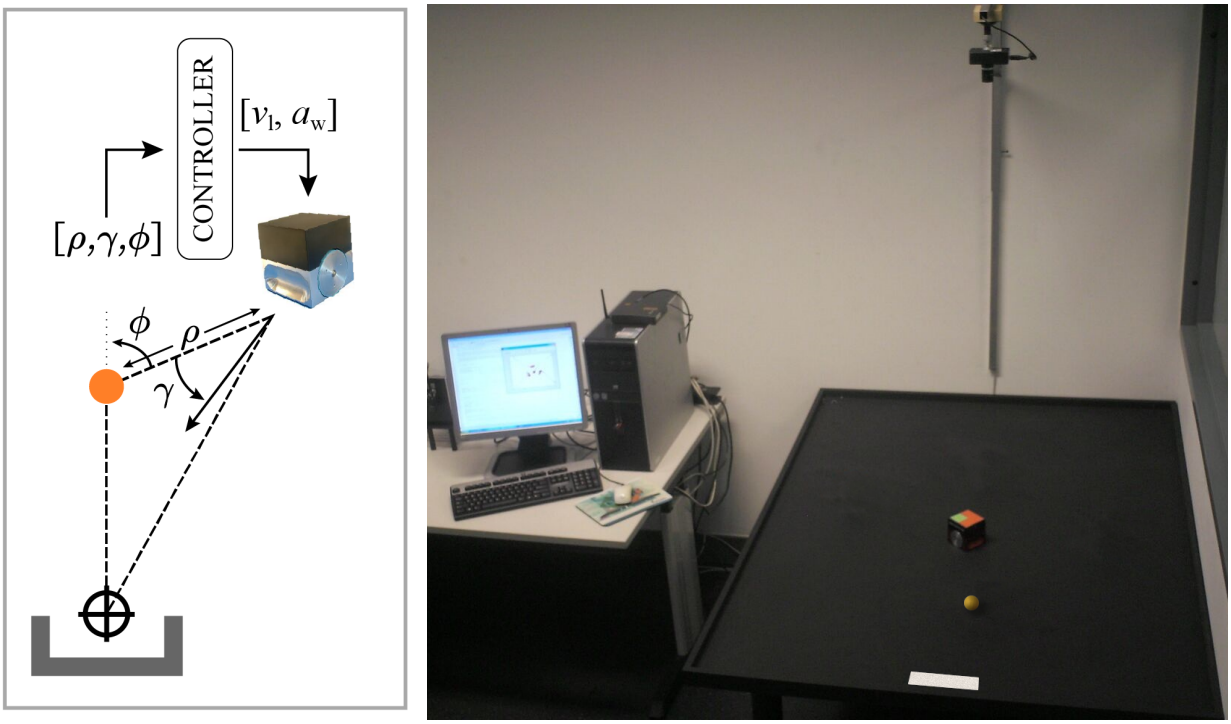

Figure 5.2: Definition of variables for the Ball-Pushing problem (left), and, a picture of the experimental setup implemented for testing the Ball-Pushing behavior (right).

to push the ball and score a goal. The MiaRobot Pro is used for this implementation (See Figure 5.2). 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 Figure 5.2 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 $20cm$ 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.

5.3.1 Centralized Modeling

For this implementation, proposed control actions are twofold $[v_l, a_w]$, the requested linear speed and the angular acceleration, where $A^{a_w} = [positive, neutral, negative]$. The 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 5.4.

Joint state space: $S = [\rho, \gamma, \phi, v_w]^T$			
State Variable	**Min.**	**Max.**	**N.Cores**
ρ	0 mm	1000 mm	5
γ	-45deg	45deg	5
ϕ	-45deg	45deg	5
v_w	-10deg/s	10deg/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			

Table 5.4: Description of state and action spaces for the DRL modeling of the Ball-Pushing problem

5.3.2 Decentralized Modeling

Stage 4.1 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, this decentralized modeling considers two individual agents for learning v_l and a_w in parallel as is shown in Table 5.4.

Stage 4.2 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 $agent_{a_w}$.

Stage 4.3 Defining the reward functions: a common reward function is considered for both CRL and DRL implementations, as is shown in Expression (5.1), where *max* features are normalization values taken from Table 5.4.

$$R(s) = \begin{cases} +1 & \text{if goal} \\ -(\rho/\rho_{max} + \gamma/\gamma_{max} + \phi/\phi_{max}) & \text{otherwise} \end{cases} \tag{5.1}$$

Stage 4.4 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. This particular modeling is not fully decentralized, however, it would be possible if $agent_{vl}$ ignores v_w in its state vector.

Stage 4.5 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, a hybrid DRL scheme (DRL-Hybrid) with a Fuzzy Q-Learning (FQL) to learn v_l is implemented, in parallel with a SARSA(λ) algorithm to learn a_w. This is a good example for depicting Stage 4.5 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, the following schemes for the Ball-Pushing problem are implemented: CRL, DRL-Ind, DRL-CAdec/CAinc/Lenient, and DRL-Hybrid. Please see Table 5.1 for the full list of acronyms. Other details about Stage 4.5 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.

5.3.3 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: $\%ofScoredGoals = scoredGoals/Episode$, 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.

5.3.4 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: $\tau = \tau_0 \exp(-\mathrm{dec} \cdot episode/maxEpisodes)$, where $episode$ is the current episode index and $maxEpisodes = 1,000$ 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 5.1. 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 $\gamma = 0.99$.

5.3.5 Physical setup

An experimental setup is implemented in order to test learned policies onto a physical setup, which is shown in Figure 5.2 (right). The Miabot Pro is connected wirelessly to a central computer close to the robot soccer platform which measures $1.5m \times 1m$. 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.

5.3.6 Results and analysis

Figure 5.3 presents learning evolution plots and Table 5.5 shows the best policy final performances. All the DRL systems implemented improved the $\%ofScoredGoals$ of the centralized one as in the learning evolution traces (Figure 5.3), as well as in the final performance test (Table 5.5). 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, and their heterogeneous action spaces make cooperation more difficult. 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 3 and described in Stage 4.5. 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 5.3, 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 5.4), which implies an exponential increase in the search space that could increase learning time, thus affecting the final performance since only $1,000$ 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

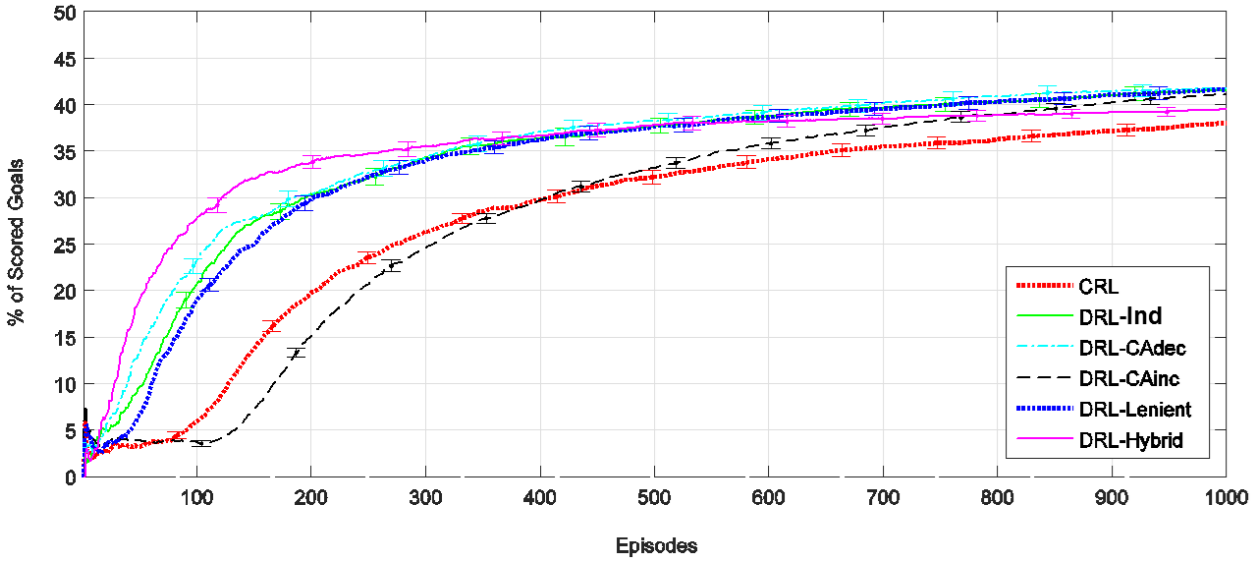

Figure 5.3: Ball-pushing learning evolution plots. Results are averaged across 25 learning runs and error bars show the standard error.

modelings or configurations can be implemented per agent. Additionally, in order to perform a fair comparison of computing time, it is also carried out a second evaluation, implementing and testing a DRL system with $N^{v_l} = 5$ actions. Once again, simulation times are measured and action-selection+Q-function-update times, obtaining 59.63s 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 5.5, 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 5.5 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 [64]. 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 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.

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

Table 5.5: Ball-pushing best policy final performances for simulation and physical robot experiments (in which 100% is the optimal case)

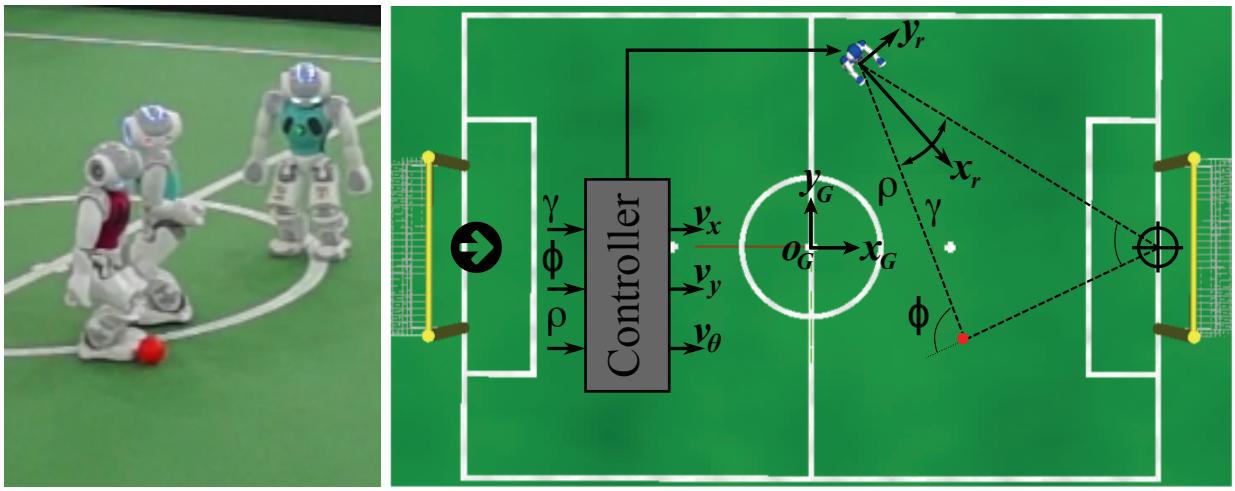

Figure 5.4: A picture of the NAO robot dribbling during a RoboCup SPL game (left), and definition of variables for dribbling modeling (right).

5.4 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. Figure 5.4 (left) shows the RoboCup SPL soccer environment where the NAO humanoid robot [17] is used. As proposed by Leottau *et al.* [29], 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 Figure 5.4 (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.

5.4.1 Proposed Decentralized Modeling

Stage 4.1 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 4.2 Identifying common and individual goals: the 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 4.3 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:

$$
\begin{aligned}
r^x &= \begin{cases} 1 & \text{if } \rho < \rho_{th} \wedge \gamma < \gamma_{th} \wedge \phi < \phi_{th} \wedge v_x \geq v_{x.max'} \\ -1 & \text{otherwise} \end{cases} \\
r^y &= \begin{cases} 1 & \text{if } \gamma < \gamma_{th}/3 \wedge \phi < \phi_{th}/3 \\ -1 & \text{otherwise} \end{cases} \\
r^\theta &= \begin{cases} 1 & \text{if } \gamma < \gamma_{th}/3 \wedge \phi < \phi_{th}/3 \\ -1 & \text{otherwise} \end{cases}
\end{aligned}
\tag{5.2}
$$

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: $[\rho_{th}, \gamma_{th}, \phi_{th}] = [250mm, 15°, 15°]$, and $v_{x.max'} = 0.9v_{x.max}$. 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 4.4 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 5.6.

5.4.2 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

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 $^\circ/s^2$	45 $^\circ/s^2$	17

Table 5.6: Description of state and action spaces for the DRL modeling of the Ball-Dribbling problem

though I 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 (5.2), but using $\gamma_{th}/3$ and $\phi_{th}/3$.

Stage 4.5 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 Figure 5.4 (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 $6x4$ meters. In order to compare the 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 [34]. 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.* [29], 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 [71] in the RoboCup [65] Standard Platform League (SPL) soccer competition.

- eRL-FLC proposed by Leottau *et al.* [35], 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, the following schemes for the Ball-Dribbling problem are implemented: DRL-Ind, DRL-CAdec/CAinc/Lenient, DRL+Transfer, CRL, RL-FLC, and eRL-FLC. Please see Table 5.1 for the full list of acronyms. Other details about Stage 4.5 are detailed in the next two subsections.

5.4.3 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.

5.4.4 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 $maxEpisodes = 2,000$ 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 4.5 Completing RL single modelings*. For all the experiments $\gamma = 0.99$.

5.4.5 Results and Analysis

Figure 5.5 shows learning evolution plots for: an independent decentralized learners scheme, DRL-Ind; a Decremental Cooperative Adaptive scheme, DRL-CAdec; DRL-CAinc; a lenient DRL implementation; and, a DRL system accelerated with transfer knowledge, DRL+Transfer, and a centralized scheme, CRL. The DRL-CAinc implementation was not able to achieve asymptotic convergence before the trained episodes. The *% of maximum forward speed* is zero during all the episodes because the robot was not able to finish successfully any episode. It is an issue about the weakness of this approach with heterogeneous actions spaces, which will be discussed at the end of this chapter.

Table 5.7 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 [34]), 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 [34], in which this dribbling problem was originally proposed based on a DRL modeling. Note 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.* [35]. 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 5.7). 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 [28]. Currently the learned policy is transferred directly to the physical robots, thus, the final performance is dependent on how realistic the simulation platform is.

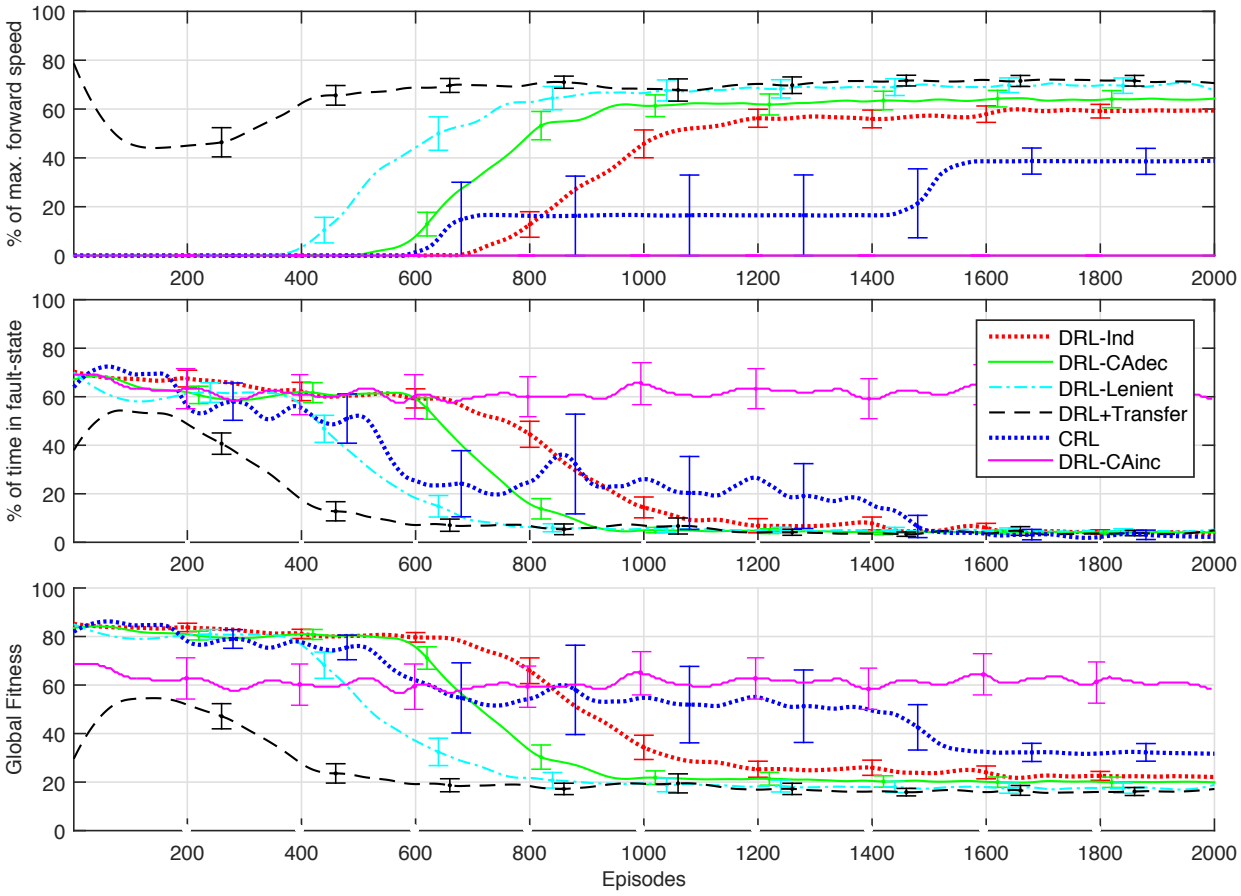

Figure 5.5: Ball-dribbling learning evolution plots.

Approach	**Performance(%)**	**Episodes to 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	61.34	†
CRL	34.93	1419

† It did not achieve asymptotic convergence before the 2,000 trained episodes, because of that its learning evolution plot is neither included in Fig. 5.5

Table 5.7: Ball-Dribbling performances (in which lower %s are better performances)

5.5 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 [36, 12]. This problem has been selected because it is one of the first DRL applications reported by Martin and De Lope [40]. Its simulation implementation is available online at Martin's repository [39] 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 Figure 5.6. The Denavit-Hartenberg parameters, the direct kinematic matrix, and more implementation details can be checked in the paper of Martin and De Lope [40], and the source files can be downloaded from Martin's repository [39].

5.5.1 Decentralized Modeling

Stage 4.1 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 $5^4 = 625$

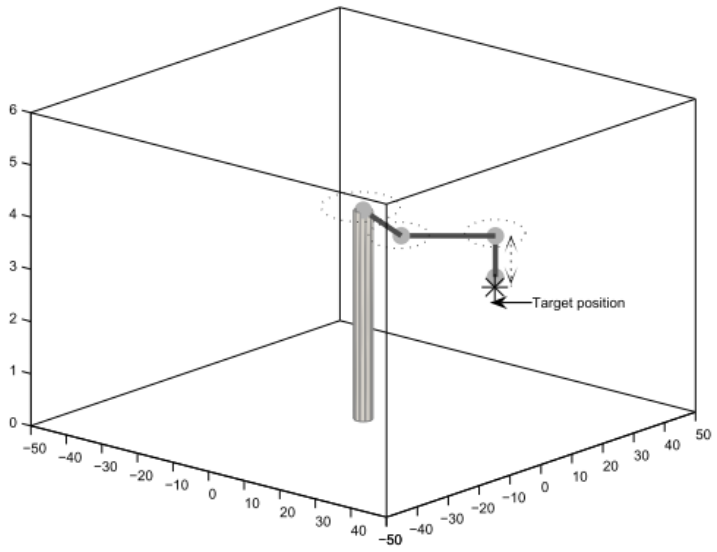

Figure 5.6: The SCARA robotic manipulator (Figure adopted from Martin and De Lope [40]).

discrete actions is computationally expensive to our current resources and purposes.

Stages 4.2 and 4.3 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 (5.3), where e$Dist$ 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 [39], this Euclidean distance based and continuous reward function is detailed and its effectively is validated.

$$
\begin{aligned}
R(s) &= \begin{cases} 10^8/(1+\mathrm{e}Dist^2) & \text{if } \mathrm{e}Dist \leq 1 \ \ \& \ penal = 1 \\ -penal \cdot \mathrm{e}Dist^5/10^4 & \text{otherwise} \end{cases} \\
penal &= \begin{cases} 10 + 0.1\theta_2 & \text{if } \theta_2 < 0 \\ 1 & \text{otherwise} \end{cases}
\end{aligned} \tag{5.3}
$$

Stage 4.4 Determining if the problem is fully decentralizable: three state variables compose the *joint state vector* $S = [\mathrm{e}_x, \mathrm{e}_y, \mathrm{e}_z]$, the error between the current end-effector position with respect to the 3-Dimensional goal position point $[x_g, y_g, z_g]$, so this modelling is not fully-decentralized. Each state variable considers three values $[-1, 0, 1]$, depending if the error is negative, near to zero, or positive.

Stage 4.5 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. 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, the following schemes for the SCARA-RTG problem are implemented: DRL-Ind, DRL-CAdec/CAinc/Lenient. Please see Table 5.1 for the full list of acronyms. Other details about Stage 4.5 are detailed in the next two subsections.

5.5.2 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.

5.5.3 RL Algorithm and Optimized Parameters

As Martin and De Lope [40] 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 $\gamma = 1$.

5.5.4 Results and Analysis

Figure 5.7 shows learning evolution plots and Table 5.8 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 [40]. 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.

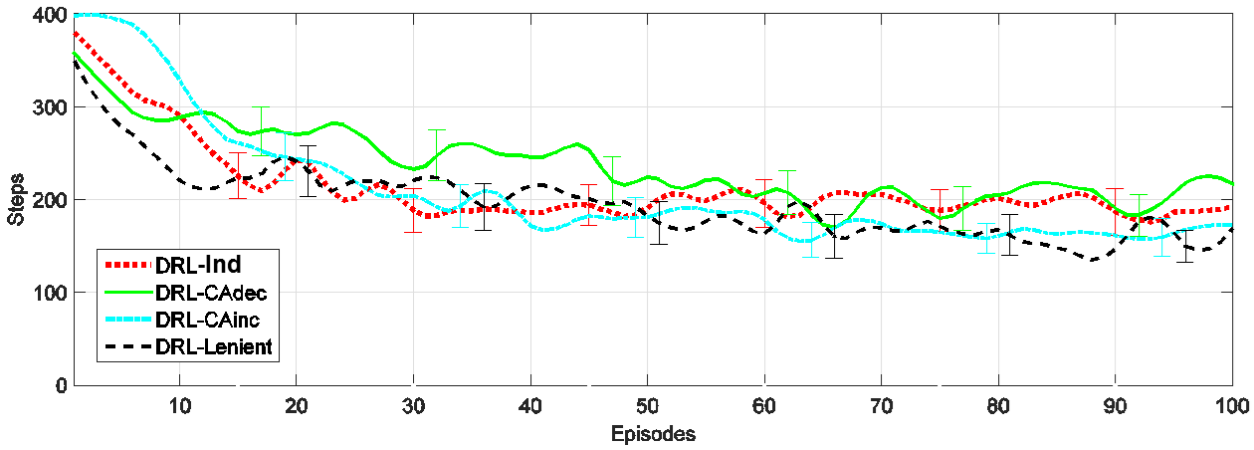

Figure 5.7: SCARA-RTG learning evolution plots.

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

Table 5.8: SCARA-RTG performances (these improve toward zero)

5.6 Discussion

One of the specific goals of this thesis was to demonstrate empirically that an independent DRL system is able to achieve faster learning times (because of the split action spaces) 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 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 non-feasible because the CRL scheme does not support incomplete observations. All the same, as is demonstrated in Subsection 5.2.5, the 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 3.4.3 and discussed below. In short, thirteen different DRL im-

plementations 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.

Most of the convergence proofs of single-agent RL algorithms assume a learning time tending to infinite, while the whole state-action space is explored. Thus, it is expected that CRL systems be able to converge under the same assumptions, achieving similar or even higher performances regarding their DRL counterparts. However, for real world problems and practical implementations with limited training, logistic and computational resources, this empirical study has evidenced that DRL systems are a feasible and effective solution to achieve asymptotic convergence faster than CRL systems.

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 5.2, this 3DMC problem with ObsL presents different state spaces, decentralized action spaces, and individual reward functions. So, by using the 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.

Table 5.9: 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

The CAinc approach was one of the algorithms with best results for the 3DMC and SCARA-RTG problems. However CAinc was not able to converge for the ball-dribbling either the ball-pushing problems, both of them implemented with heterogeneous and non-symmetrical sized action spaces, contrary to the modelings of 3DMC and SCARA-RTG cases. Thus, according the obtained experimental results, it can be preliminary said that this algorithm is only effective for DRL problems modelled with homogeneous or symmetrical sized action spaces. Of course, this is an empirical hypothesis which must be validated more rigorously. This is a non-trivial limitation which will be a focus for future research.

Lenient and CA algorithms have evidenced the best averaged final performances for the four tested problems. At least one of 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 5.9. 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 5.9 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 Figures 5.1-5.7 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, a unique set of RL parameters for each of the DRL imple-

mented problems is used. If individual sets of parameters per each individual agent were optimized, results may have been outperformed for those DRL schemes in which the agents are heterogeneous such as ball-pushing and dribbling.

5.7 Summary

This chapter addressed the DRL of individual behaviors of those problems in which multi-dimensional action spaces are involved. The DRL methodology introduced in Chapter 4, as well as the DRL-Ind, DRL-Lenient, and DRL-CA 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 or DRL-CA MAL algorithms showed the best final performances for the four tested problems.

Chapter 6

Accelerating Decentralized Reinforcement Learning

This chapter proposes the use of Knowledge Transfer (KT) to coordinate and accelerate the DRL of complex individual robot behaviors. Relevant background and related work are presented in Section 6.1. Section 6.2 provides a literature review. Two KT based DRL schemes are described and implemented in Section 6.3: DRL+CoSh, a DRL scheme accelerated-coordinated by using the Control Sharing KT approach; and DRL+NeASh, the Nearby Action Sharing approach proposed for including a measure of uncertainty to the CoSh procedure. These proposed schemes are validated through two complex real-world problems: the inwalk-kicking and the ball-dribbling behaviors, which are described and analyzed in Section 6.4. Finally, results obtained are discussed in Section 6.5.

This chapter is fully based on our paper [30]: *Accelerating Decentralized Reinforcement Learning of Complex Individual Behaviors*, which has been submitted to the journal: *Engineering Applications of Artificial Intelligence*.

6.1 DRL and Transfer Knowledge Overview

KT is used to accelerate the rate at which one or more target tasks are learned from one or more sources of knowledge. Two reasonable goals of KT are: *(i)* to effectively reuse past knowledge in a novel task, and *(ii)* to reduce the overall time required to learn a complex task. In the context of DRL-Ind, a third goal will be considered: to address the coordination problem.

Most of the stochastic DRL systems implemented with independent learners present two main drawbacks: non-stationary and non-Markovian issues. Laurent et al. [26] indicate that these drawbacks could be mitigated by: *(i)* decaying the exploration rate, and *(ii)* using coordinated exploration techniques for shrinking the action space. Both mechanisms can be accomplished by using KT approaches [23, 1]. In fact, decayed exploration and transfer rates are commonly considered parameters in some KT approaches like [23, 1]. KT also

allows shrinking the action space by exploring in a subset of actions limited by a source of knowledge (SoK), which has been previously learned or designed. A SoK for a DRL system should contain at least one branch per decentralized learning agent. If those branches are pre-coordinated, the SoK relieves at the same time the coordination problem.

Different cases of KT applied to DRL tasks can be identified: same tasks (source and target) and same problem-spaces; same tasks and different problem-spaces; different tasks and same problem-spaces; and different tasks and different problem-spaces. The problem-spaces refer to the source and target state variables and actions. In addition, DRL systems consider the case of homogeneous and heterogeneous agents, in which homogeneous agents have identical problem-spaces and goals.

For this work, the following cases are considered:

- Heterogeneous agents, in order to allow designing an individual goal for each independent learner.
- Different source and target tasks, in order to allow the use of layered learning [32] and easy mission approaches [55].
- The same source and target problem-space, in order to avoid source-task-selection or task-mapping [58], which may slow-down and complicate the procedure.

6.2 Literature Review

KT has been widely studied and applied to accelerate single-agent RL [58]. To a lesser extent, KT has been used for MARL systems as well, and not only to accelerate but also to outperform and address large-scale problems [67, 1]. Since to the best of our knowledge, this is the first work reporting a DRL implementation accelerated-coordinated by using KT. Thus, this section overview relevant and similar works but in the context of KT applied to MARL.

Yujing Hu et al. [72] present the idea of equilibrium transfer based MARL, which outperforms and accelerates considerably its non-transfer counterpart, and scales significantly better than algorithms without equilibrium transfer when the state/action space grow and the number of agents increases. Compared to the KT methods used in this work, this method only has been validated on discrete domains, and several non-trivial considerations must be taken into account for implementing effectively the transfer approach. However this method seems an interesting alternative for future DRL implementations. Bianchi *et al.* [1] present the heuristically accelerated MARL (HAMRL) algorithms, as a general framework for including heuristic functions to influence the action choice of the agents. This family of algorithms must be modified and adapted depending on the source and target MARL algorithm to be used; however, it is also an interesting approach for future implementations, since CoSh can be seen as a simplified version of HAMRL. Vrancx et al. [67] apply transfer learning to the Coordinating Q-learning framework, which uses a statistical test to sample the immediate rewards received by the agent. This approach shows interesting results and it is worth considering it for future and more sophisticated DRL implementations. By contrast to the KT

methods used in this work, this algorithm uses previously identified problem states as samples to train a rule based classifier, which makes somehow complex a fast implementation. Taylor et al. [57] propose a parallel transfer learning for MAS which, unlike to the approach here-proposed, is able to learn simultaneously the source and target tasks, and share their current experience based on a set of rules and considerations somehow focused on the particular case studied, a smart grid. Boutsioukis et al. [4] apply transfer learning in MARL domains, showing that the transfer method reduces the learning time and increases the asymptotic performance. Similar to the KT-DRL approaches used in this work, this method biases the initial action value function, but it is only validated in a discrete, deterministic and 2-agents competitive domain.

6.3 Proposed KT-based DRL

This chapter proposes to use KT to help DRL-Ind in coordinating the exploration during the early episodes. Under this approach, a subset of actions taken from a prior-coordinated SoK is used for guiding the agents to avoid unknown actions, while they evolve leniently and the non-Markovian effects are reduced. The SoK acts as an initial value function, and thereby endows the agent with an initial policy [24]. If both decayed exploration rate and decayed probability of KT are used, the subset of actions from the SoK is progressively increased or modified over time, while concurrent exploration is reduced. In this way, each agent finds easily the best response to the behavior of the others.

Several requirements are taking into account in order to choose the KT strategy used in this work. In general, this work considers methods that are able to:

1. Transfer toward any RL algorithm that uses an action value function, such as the works reported in [60, 23, 41, 4, 1].

2. Transfer from different SoK types such as standard controllers (e.g. linear or fuzzy controllers), hand-craft behaviors, and RL policies such as the works reported in [15, 23, 1].

3. Scale-up when the state/action space grow and the number of agents increases, but without extra memory consumption, such as [67, 23, 1]

4. Avoid to build empirical or on-line models of the other agents strategies such as [23, 1].

5. Avoid to consider other agents' actions nor action choice negotiation mechanisms such as [23, 72].

Note that the CoSh approach [23] accomplishes all the listed considerations. In addition, CoSh also supports the KT case already indicated in Section 6.1: heterogeneous agents, different source and target tasks, and the same source and target problem-space.

6.3.1 Control Sharing (CoSh)

Introduced by Knox and Stone [23], CoSh acts only during action-selection, without affecting the updates of the Action-Value functions. This method effectively either lets the RL agent to choose its action or takes a_{src}, the action shared from the SoK. If an action is shared, the RL agent observes and updates it as if it were making the choice. The action a is chosen by SoK or *source-policy* ($\pi_{src}(s) = a_{src}$) with probability β as

$$P(a = a_{src}) = min(\beta, 1), \tag{6.1}$$

otherwise action a is chosen using a base RL agent's action-selection mechanism. β is decayed periodically by a predefined factor.

CoSh was originally proposed for the single-agent RL case, but it can be easily extended to the DRL case if a SoK is available to each separate agent and if decayed and synchronized exploration rates are implemented as in Algorithm 3.1, as well as with the transfer probabilities are described as below. Note that CoSh is able to accomplish all the requirements previously mentioned precisely because its extreme simplicity. This is an advantage in order to quick implementations because only the decayed probability β must be set. However, in the next subsection, I am proposing the Nearby Action Sharing variant for continuous action spaces, which also fulfills those requirements and additionally is able to include a measure of uncertainty to the transferred action for noisy SoKs.

6.3.2 Nearby Action Sharing (NeASh)

Same as CoSh, NeASh also acts only during action-selection, transferring knowledge from continuous action spaces, when no information different to the suggested action in an observed state is available from the SoK. NeASh has applicability in cases where the sources of knowledge are standard controllers, hand-coded behaviors, rule inference systems, among other similar sources. NeASh is based on CoSh, but it takes advantage of continuous action spaces to compensate the lack of information or uncertainty about the quality of the source actions. It assumes that a measure of the quality of a state-action pair is related to its distance to the action a_{src} suggested by the SoK (e.g., a source policy $\pi_{src}(s)$). In this way, a normal distribution along the universe of discourse centered in a_{src} is considered (see Figure 6.1), and the resulting nearby action to share is $a'_{src} = \mathcal{N}(\mu, \sigma)$, in which $\mathcal{N}(\mu, \sigma)$ is a normally distributed random generator with mean $\mu = a_{src}$ and standard deviation $\sigma = \varrho(1-\beta)$. Algorithm 6.1 depicts the procedure for a DRL system accelerated-coordinated by using NeASh, which has M single-agents.

As in the CoSh case, the action is chosen by *source-policy* with probability β. Typically, the initial value of β is 1. β is decayed periodically as well as the standard deviation of $\mathcal{N}$, which means that, at the beginning of the learning process, NeASh works similarly to CoSh in Equation (6.1). However, while the learning process goes along, the probability of choosing an action a'_{src} increasingly goes away from the action a_{src} as $\varrho(1-\beta)$. Actually, NeASh turns into CoSh for the particular case of $\varrho = 0$.

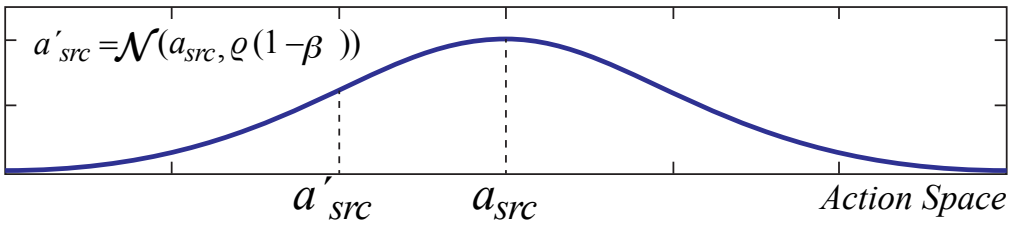

Figure 6.1: Normal random function proposed to NeASh approach.

If no KT is selected during a step, as in the case of line 6.1.22, then NeASh offers the option of using its own action-selection mechanism or just using a regular approach (e.g., ε-greedy) as depicted in line 6.1.28. The NeASh action-selection mechanism works similar to Softmax [53], but taking advantage of the continuous action spaces, and uses a normal distribution instead of a Boltzmann one. In a very similar way to obtaining a'_{src}, the chosen action from the target policy a'_{tgt} is obtained by using the best action from the current target policy (e.g., $\max Q^m(s^m)$), but with $\sigma = \varrho\beta$, and taking a nearby target action as in line 6.1.26.

A synchronized transfer/exploration version of NeASh can be implemented by using a unique random number as in Lines 6.1.8-6.1.9, instead of M different random numbers for synchronizing transfer/exploration such as in Lines 6.1.11-6.1.13. Note that if normal distributions are used, it is necessary to bound a'_{src} and a'_{tgt} into the action space with module or clip functions. This issue can be solved by using other finite support kernels such as triangular, cosine or Epanechnikov [38].

6.4 Experimental Validation

In order to validate MAS benefits and properties of the DRL systems coordinated-accelerated by using KT, three different schemes are implemented in this experimental validation:DRL+CoSh, DRL+NeASh, and the DRL-Ind with a decayed exploration rate, which has been introduced in Section 3.4.1. These three schemes are validated through two complex and real-world problems from soccer robotics: the *inwalk-kicking* and the *ball-dribbling*. Both behaviors are performed with humanoid biped robots, which results very challenging because their modeling must take into account the physical interaction between the ball, the ground, the robot's feet, and the robot's gait inertia. Thus, the action is highly dynamic, non-linear, and influenced by several sources of uncertainty. Moreover, these two behaviors are actually used by the UChile Robotics Team [70] in the RoboCup [65] Standard Platform League (SPL) soccer competition, in which the team has been regular semifinalist in the recent world competitions.

The inwalk-kicking and ball-dribbling are part of the *inwalk-ball-pushing based behaviors* proposed in [34]. Similar to [29], the description of both problems use the following variables: $[v_x, v_y, v_\theta]$, the velocity vector; γ, the robot-ball angle; ρ, the robot-ball distance; ψ, the ball-target distance; and ϕ, the robot-ball-target complementary angle. These variables are shown in Figure 6.2, where the desired target $\oplus$ is the opponent's goal, and a robot's egocentric reference system is indicated with the x axis pointing forwards. Figure 6.2 also shows the

Algorithm 6.1 DRL+NeASh: KT and action selection mechanism

Parameters:

M ▷ Number of decentralized learning agents

ϱ^m ▷ Scale factor for the continuous action space of $agent_m$, where $m = 1, \cdots, M$, $\varrho \geq 0$

β ▷ Probability of choosing the action from π_{src}. β is periodically decayed $\in [0, 1]$

Inputs:

$S^1, \cdots, S^M$ ▷ State space of each agent

$A^1, \cdots, A^M$ ▷ Action space of each agent

repeat for each step:

  if Synchronized exploration and transfer **then**

    $urnd \leftarrow$ a uniform random variable $\in [0, 1]$

    $\mathcal{N}_T, \mathcal{N}_E \leftarrow$ normal random variables ($\mu = 0, \sigma = 1$) for transferring and exploration procedures respectively

  else

    $[nrnd^1, \cdots, urnd^M] \leftarrow$ a uniform random vector $\in [0, 1]$

    $[\mathcal{N}_T^1, \cdots, \mathcal{N}_T^M] \leftarrow$ a normal random vector ($\mu = 0, \sigma = 1$)

    $[\mathcal{N}_E^1, \cdots, \mathcal{N}_E^M] \leftarrow$ a normal random vector ($\mu = 0, \sigma = 1$)

  end if

  for all agent $m \in M$ **do**

    $s^m \leftarrow$ Get state of $agent_m$

    if $urnd^m < \beta$ **then**

      $a_{src}^m \leftarrow \pi_{src}(s^m)$ Get the action from the source-policy for $agent_m$

      $\mu \mathrel{+}= a_{src}^m$

      $\sigma \mathrel{*}= \varrho^m(1 - \beta)$

      $a^m \leftarrow a_{src}^{'m} = \mathcal{N}_T^m(\mu, \sigma)$

    else if NeASh action-selection mechanism is used **then**

      $a_{tgt}^m \leftarrow$ Get the best action from the current target policy of RL $agent_m$

      $\mu \mathrel{+}= a_{tgt}^m$

      $\sigma \mathrel{*}= \varrho^m \beta$

      $a^m \leftarrow a_{tgt}^{'m} = \mathcal{N}_E^m(\mu, \sigma)$

    else

      $a^m \leftarrow$ Set action from the current RL action selection mechanism of $agent_m$

    end if

  end for

until Terminal condition

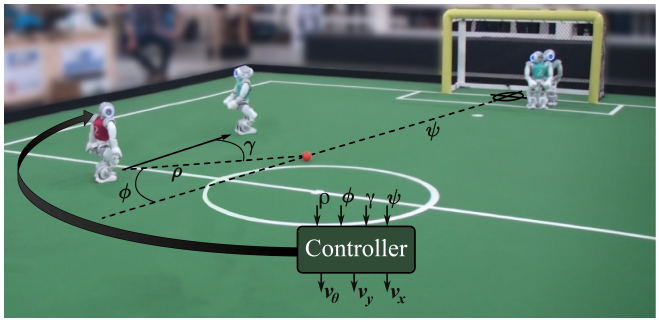

Figure 6.2: Geometric state variables and control actions for the ball-pushing based behaviors, performed by the NAO robot using a magenta jersey in a real RoboCup game.

RoboCup SPL soccer environment where the NAO humanoid robot [17] is used.

A description of each problem as well as the implementation and modeling details are presented in the next sub-sections. The experimental results are then discussed, for which we use the following terminology: DRL-Ind is an independent learners scheme implemented without any kind of MA coordination; DRL+CoSh and DRL+NeASh are DRL schemes accelerated using CoSh and NeASh transfer knowledge approaches, respectively; RL-FLC is an implementation reported in [29, 35], which combines a Fuzzy Logic Controller (FLC) and an RL single agent. For the kicking problem, some extra experiment are carried out by using different sources of knowledge for CoSh and NeASh transfer methods, namely SrcHQ and SrcLQ, a high and a low quality sources, respectively. This is explained in Section 6.4.1. All the acronyms of the implemented methods and problems are listed in Table 6.1.

Acronym	**Algorithm's parameters**
inwalk-Kicking	
2D Simulator	
DRL-Ind	$\varepsilon_0 = 1,\ dec = 30,\ \alpha = 0.2,\ \lambda = 0.9,\ \gamma = 0.99$
DRL+NeASh-SrcHQ	$\varrho = 25,\ \beta = 1,\ dec = 29,\ \alpha = 0.2,\ \lambda = 0.9,\ \gamma = 0.99$
DRL+NeASh-SrcLQ	$\varrho = 16,\ \beta = 1,\ dec = 12,\ \alpha = 0.2,\ \lambda = 0.9,\ \gamma = 0.99$
DRL+CoSh-SrcHQ	$\varepsilon_0 = 1,\ \beta = 1,\ dec = 19,\ \alpha = 0.2,\ \lambda = 0.9,\ \gamma = 0.99$
DRL+CoSh-SrcLQ	$\varepsilon_0 = 1,\ \beta = 1,\ dec = 13,\ \alpha = 0.2,\ \lambda = 0.9,\ \gamma = 0.99$
3D Realistic Simulator	
DRL-Ind	$\varepsilon_0 = 1,\ dec = 30,\ \alpha = 0.2,\ \lambda = 0.9,\ \gamma = 0.99$
DRL+NeASh	$\varrho = 10,\ \beta = 1,\ dec = 15,\ \alpha = 0.2,\ \lambda = 0.9,\ \gamma = 0.99$
DRL+CoSh	$\varepsilon_0 = 1,\ \beta = 1,\ dec = 15,\ \alpha = 0.2,\ \lambda = 0.9,\ \gamma = 0.99$
Ball-Dribbling	
3D Realistic Simulator	
DRL-Ind	$\varepsilon_0 = 1,\ dec = 20,\ \alpha = 0.2,\ \lambda = 0.9,\ \gamma = 0.99$
DRL+NeASh	$\varrho = 5,\ \beta = 0.5,\ dec = 10,\ \alpha = 0.2,\ \lambda = 0.9,\ \gamma = 0.99$
DRL+CoSh	$\varepsilon_0 = 1,\ \beta = 1,\ dec = 30,\ \alpha = 0.2,\ \lambda = 0.9,\ \gamma = 0.99$
RL-FLC	Final performance taken from [35]

Table 6.1: Experiment's acronyms and their optimized parameters

6.4.1 Inwalk Kicking

In the inwalk-kicking behavior, a robot attempts to shoot and score a goal by performing an *inwalk-ball-pushing* [38]. A general version of this behavior was originally introduced and implemented in [54], in which the gait phases are modified to create kick motions. The proposed implementation consists of inwalk kicks using only the inertia of the gait, without any specially designed kick motion. The robot just push the ball as hard as possible while it is walking toward the ball, as proposed in [38].

Decentralized Modeling

Since the velocity vector of the biped-robot walking-engine is $[v_x, v_y, v_\theta]$, it is possible to decentralize this 3-Dimensional action space by using three separate agents, namely $Agent_x$, $Agent_y$, and $Agent_\theta$. The expected common goal is to walk fast toward the ball, and pushing it aligned and hard enough for scoring a goal. That means: to maximize v_x and minimize $\rho, \gamma, \phi, v_y, v_\theta$ before pushing the ball; and to minimize ϕ, ψ once the ball is shoot. So, the proposed control signals are $[v_x, v_y, v_\theta]$, and the proposed common reward is:

$$r = \begin{cases} K \exp\left(-\psi_{error}/\psi_0\right) \exp\left(-\alpha_{error}/\alpha_0\right), & \text{if } \textit{ball} \text{ is } \textit{push}\text{ed}, \\ -\left(\rho/\rho_{max} + |\phi|/\phi_{max} + |\gamma|/\gamma_{max}\right), & \text{otherwise}, \end{cases} \tag{6.2}$$

where $[\rho_{max}, \gamma_{max}, \phi_{max}] = [2000mm, 90°, 90°]$, ψ_{error} is the distance that the ball still needs to travel to reach the target in its current trajectory, α_{error} is the angle deviation of the ball's trajectory from a straight line to the target, and the parameters K, ψ_0 and α_0 allow the design of rewards with more focus on kick strength or precision. The complete proposed modeling for learning the 3-Dimensional velocity vector from the joint observed state is detailed in Table 6.2.

Joint state space: $S = [\rho, \gamma, \phi, \psi]^T$			
State Variable	**Min.**	**Max.**	**N.Cores**
ρ	0 mm	800 mm	15
γ	-70°	70°	11
ϕ	-90°	90°	13
Action space: $A = [v_x, v_y, v_\theta]$			
Agent	**Min.**	**Max.**	**N.Actions**
v_x	0 mm/s	120 mm/s	16
v_y	-70 mm/s	70 mm/s	15
v_θ	-30 $°/s$	30 $°/s$	17

Table 6.2: Description of state and action spaces for the DRL modeling of the inwalk-kicking problem.

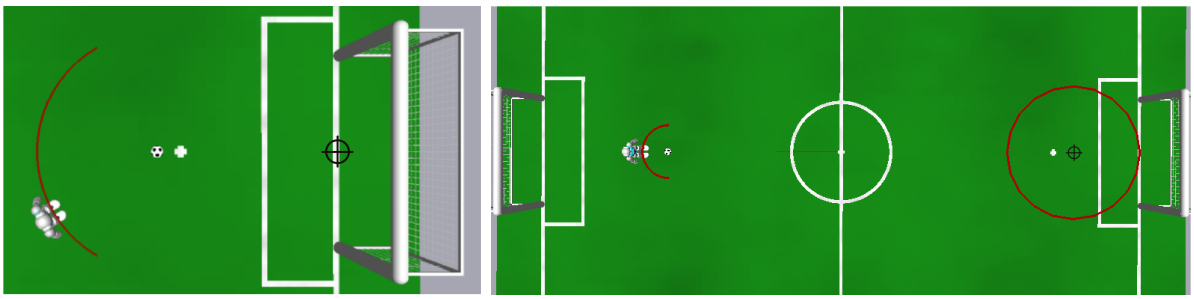

Figure 6.3: The learning setup environment of the inwalk-kicking problem (left), and the ball-dribbling problem (right).

Experimental Setup

The inwalk-kicking RL procedure is carried out episodically. After a reset, the ball is set on a fixed position, 1.5m in front of the opposite goal as shown in Figure 6.3 (left). The robot is set on a random position, 1m around the ball and always facing it. This random initialization is designed so the learning agent can successfully learn many operation points, and thus achieve a general kick behavior. The episode's termination criterion is given by the following conditions: episode timeout of 200 seconds, the robot or the ball leaves the field, or the ball is pushed by the robot.

A SARSA(λ) RL algorithm with RBF approximation is implemented for these experiments. DRL-Ind and DRL+CoSh use ε-greedy decayed as

$$\varepsilon = \varepsilon_0 \exp(-\text{dec} \cdot \text{episode}/maxEpisodes) \tag{6.3}$$

where dec is a decayed factor, *episode* is the current episode index, and $maxEpisodes = 2,000$ trained episodes per run. DRL+CoSh also decay β in the same way. DRL+NeASh uses the same decay function, but applied for annealing β and managing the knowledge transfer and the NeASh action-selection mechanism depicted in Algorithm 6.1.

The percentage of scored goals across the trained episodes is considered as the performance index

$$ScoredGoalRate(\%) = scoredGoals/EpisodeWindow \tag{6.4}$$

where $scoredGoals$ are the number of scored goals during $EpisodeWindow = 200$ episodes, a window of 10% of the 2,000 total trained episodes.

Two kinds of experiments are carried out: *(i)* an extensive experimental procedure carried out on a 2D simulator in which basic kinematics models are computed faster, allowing parameter optimizations and running several trials for more statistical significance; and *(ii)* experiments carried out in the SimRobot 3D simulator released in [54], which is very realistic but computationally expensive for the Intel(R)Core(TM)i7-4774CPU@3.40Ghz available on

our lab (a run of $2,000$ episodes may take up to 12 hours). The experimental procedure for both experiments is described below:

2D Simulator Experiments:

- Five different schemes are tested: DRL-Ind, DRL+CoSh-SrcHQ, DRL+CoSh-SrcLQ, DRL+NeASh-SrcHQ, and DRL+NeASh-SrcLQ. Since NeASh and CoSh approaches require a source for transferring knowledge, a linear controller of the form:

$$\begin{bmatrix} v_x \\ v_y \\ v_\theta \end{bmatrix} = KX = \begin{bmatrix} k_{x\rho} & \cdots & k_{x\phi} \\ \vdots & \ddots & \vdots \\ \cdot & \cdots & k_{\theta\phi} \end{bmatrix} \begin{bmatrix} \rho \\ \gamma \\ \phi \end{bmatrix} \tag{6.5}$$

  with two configurations: SrcHQ, a high quality source in which matrix $\boldsymbol{K}$ in Eq. 6.5 was tuned for the best performance achievable by these linear controllers (around 20% on average); SrcLQ, a low quality source in which $\boldsymbol{K}$ was tuned only for achieving the ball without kicking it. These two different sources of knowledge are tested in order to analyze their impact in the final performance and learning time.

- The decay factor (dec) and the action space scale factor (ϱ) parameters were optimized for each of these five implementations by using the custom hill-climbing algorithm presented in [31]. This is an important step in order to guarantee that every scheme tested uses the best parameter settings. In this way, comparisons and evaluations are carried out based on the best performance potentially achievable by each method, according to the optimization results. All the parameters are detailed in Table 6.1.

- The best set of parameters of each implemented scheme is evaluated 25 times and learning evolution plots are averaged. Final performances are measured as well as the number of episodes required to achieve a given performance, which is called *time to threshold*.

3D Realistic Simulator Experiments:

- In order to validate the 2D simulator experiments, three implementations are tested: DRL-Ind, DRL+CoSh-SrcHQ, and DRL+NeASh-SrcHQ. The averaged performance of SrcHQ for this case is around 15%.

- The decay factor (dec) and the action space scale factor (ϱ) parameters were also optimized for each of these five implementations. Since computing time limitations, in this case a manual search exploring of the whole parameter space was used for finding the best sets of parameters.

- The best set of parameters of each implemented scheme is evaluated 10 times and learning evolution plots are averaged.

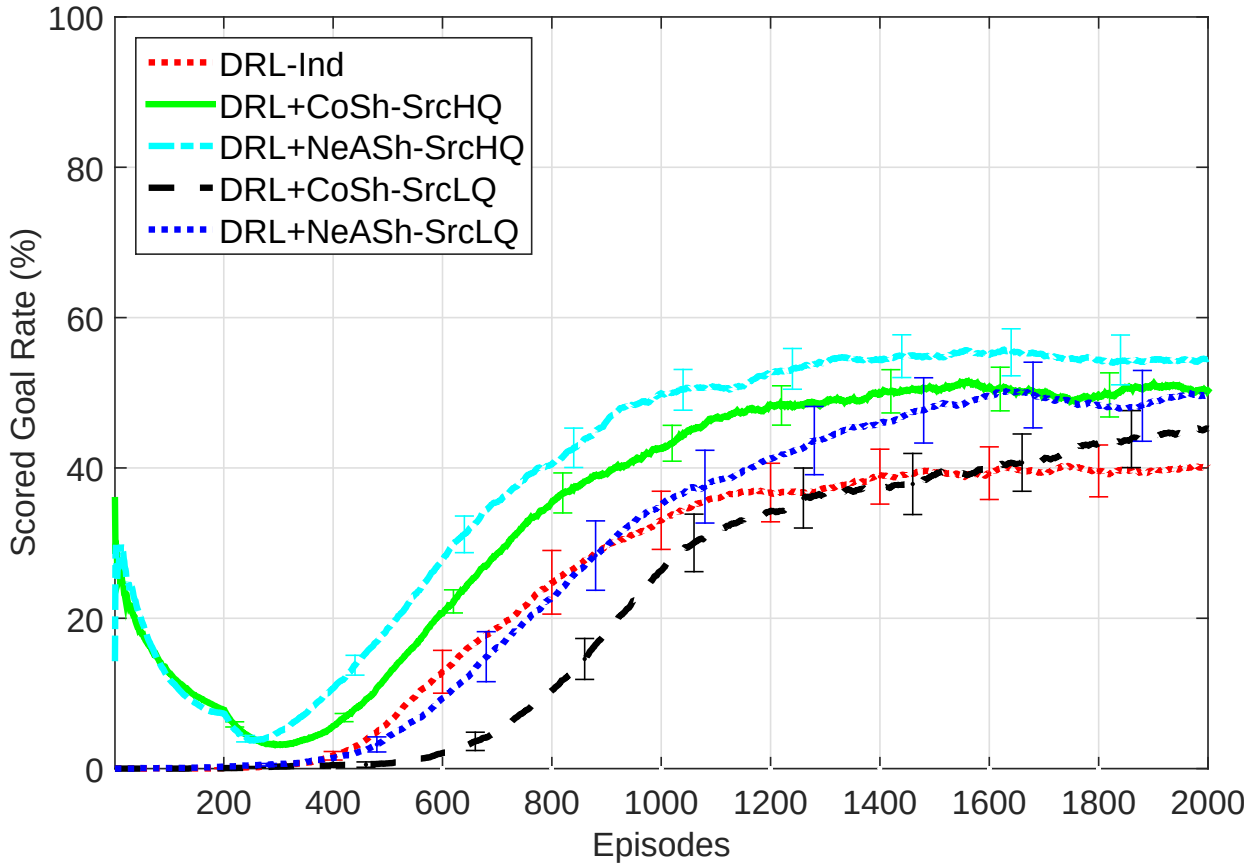

Figure 6.4: inwalk-kicking learning evolution plots with two different sources of knowledge. Results are averaged across 25 learning runs and error bars show the standard errors.

Results and Analysis

Figure 6.4 shows learning plots for 2D simulator experiments. Table 6.3 presents their final performances and learning times for achieving a time to threshold of 40%. DRL+NeASh schemes show the best final performance and learning times (around 14% better and 43% faster than DRL-Ind) followed by DRL+CoSh schemes (around 10% better and 36% faster than DRL-Ind). DRL-Ind shows the lowest performance and slower learning times, which is expected taking into account the lack of prior-coordination, unlike NeASh and CoSh approaches which use linear controllers as SoK.

Two different qualities of the SoK were tested for comparing NeASh and CoSh performances. Both transfer methods outperform the DRL-Ind even when a low quality source is used. However, note from Figure 6.4 that the DRL+CoSh-SrcLQ learning plot shows lower performance during most of the learning procedure, evidencing that the quality of the SoK affects CoSh more than NeASh. From Table 6.3, DRL+NeASh-SrcHQ is about 21% faster than DRL+CoSh-SrcLQ, and 23% faster than DRL-Ind. This can be explained taking into account the nearby action effect, because while CoSh always shares the same action for a determined state, NeASh explores the neighborhood according ϱ and $1-\beta$ for sharing a nearby action, which eventually can have a better performance, but surely gives more experience

Approach	**Final performance(%) [Scored Goal Rate]**	**Time to Th.** (40%)
2D Simulator		
DRL+NeASh-SrcHQ	54.55	775
DRL+CoSh-SrcHQ	50.28	915
DRL+NeASh-SrcLQ	49.63	1165
DRL+CoSh-SrcLQ	45.22	1592
DRL-Ind	40.16	1631
3D Simulator		
DRL+NeASh	58.53	531
DRL+CoSh	57.02	688
DRL-Ind	48.16	1064

Table 6.3: inwalk-kicking performances (in which 100% is the optimal policy).

and information to the target DRL agents. As a disadvantage, NeASh requires tuning the extra parameter ϱ.

It is interesting analyzing the effect of those parameters on the knowledge transfer. From Table 6.1, note that CoSh uses a smaller decay factor to deal with the lower quality in the source ($dec : 19 \rightarrow 13$), which implies a slower learning. NeASh reacts similarly: it reduces dec from 29 to 12, but increases the nearby action deviation by reducing the scale factor ϱ from 25 to 16. This means, if the source is good, that NeASh trusts more in the SoK, but if the source is weak, that NeASh should explore more around the suggested action from the source.

Figure 6.5 presents learning evolution plots for the 3D simulator experiments, in which the high quality sources were used. These plots validate results from previous experiments: DRL-NeASh is again the best and fastest scheme (around 10% better and 27% faster than DRL-Ind) followed by DRL+CoSh schemes (around 9% better and 19% faster than DRL-Ind). From Table 6.1, note that ϱ and dec differ from 2D experiments due to the more challenging and realistic environment. For instance, DRL-NeASh now uses $\varrho = 10$, which is a reduced value compared to the 2D experiment, in order to increase the exploration zone from the source actions.

6.4.2 Ball-Dribbling

The Ball-dribbling behavior has been introduced in Section 6.4.2. The same decentralized modeling and implementation details are considered in the following experiments.

Experimental Setup

A SARSA(λ) RL algorithm with RBF approximation is also implemented for these experiments. Parameters and decayed functions are set and configured in the same way as for the

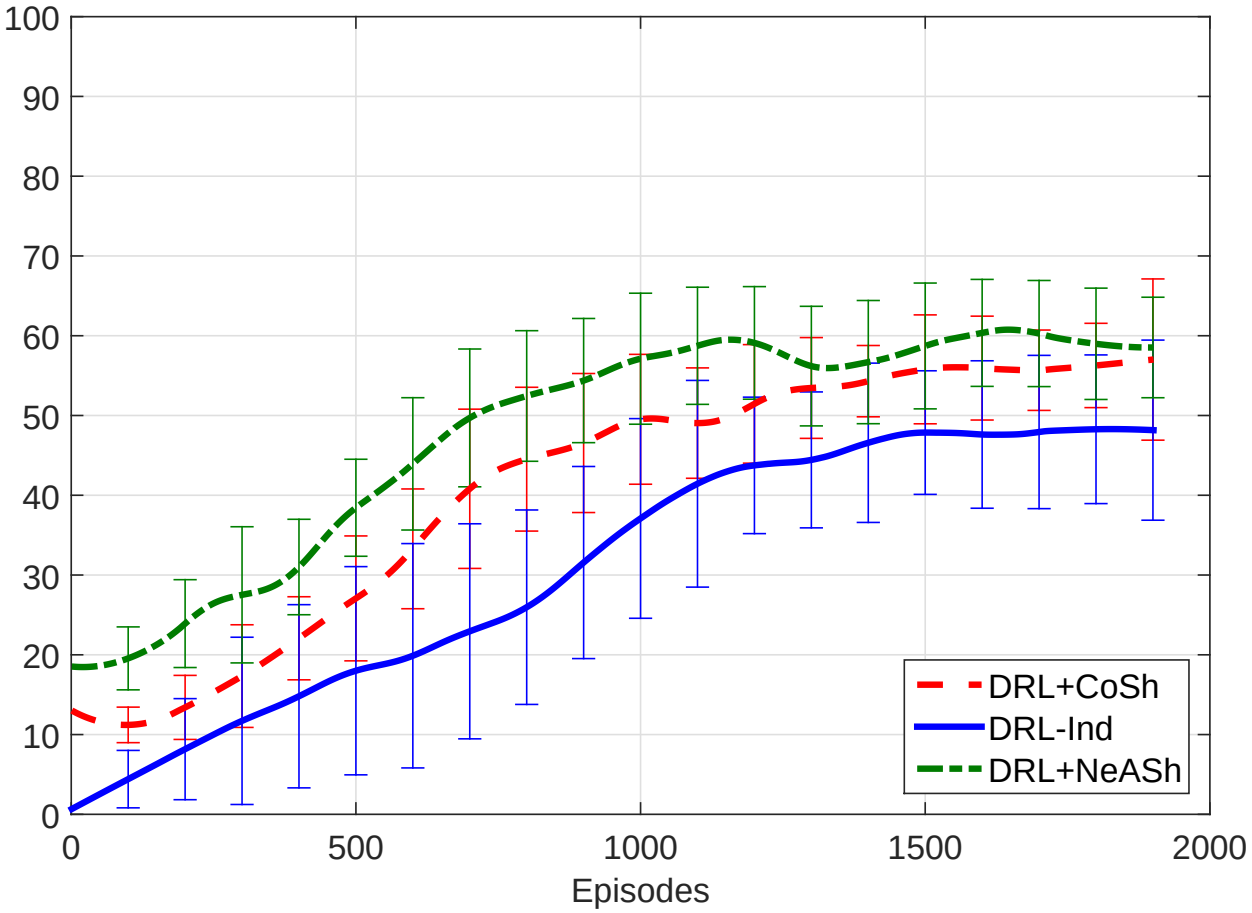

Figure 6.5: inwalk-kicking learning evolution plots for the 3D realistic simulator. Results are averaged across 10 learning runs and error bars show the standard errors.

kicking problem. All the parameter are detailed in Table 6.1 for each scheme implemented.

The ball-dribbling RL procedure is carried out episodically and $1,000$ episodes are trained in the SimRobot 3D simulator. After a reset, the robot is set near to its own goal (Figure 6.3, right), in a random position over the red arc around the ball, and the desired target is defined by $\oplus$. The terminal state is reached if the robot loses the ball, if the robot leaves the field, or if the robot reaches the target (which is the expected terminal state). The training field is 9×6 meters.

The evolution of the learning process is evaluated by measuring and averaging 10 runs. The *Global Fitness* proposed in Section 5.4.3 is considered here as performance index.

Results and Analysis

Figure 6.6 shows the learning evolution plots and Table 6.4 shows the averaged final performances and learning times for achieving a performance of 30%. DRL+NeASh scheme shows the best final performance and learning times, around 7% better and 62% faster than DRL-Ind, followed by DRL+CoSh scheme which is around 6% better and 58% faster than DRL-Ind. DRL-Ind shows the lowest performance, slower learning times, and larger error bars with respect to the TK approaches. Same as in the inwalk-kicking problem, it is expected due to the lack of prior-coordination in the DRL-Ind scheme, contrary to the DRL+CoSh and DRL+NeASh schemes, which for this experiments used a SoK with a prior performance of around 45%. Note from Table 6.1, that after DRL+NeASh's parameter optimization, $\beta = 0.5$. It is expected because the controlled exploration of the NeASh approach, which needs less KT from the SoK. Because of that, its initialization showed in Figure 6.6, differs from DRL+CoSh.

Since a previous implementation for the ball-dribbling problem has been already reported in the literature as RL-FLC [29], its performance indices are included in Table 6.4. The effectiveness and benefits of this hybrid RL and fuzzy approach have been pointed out in [29]. However, a significant human effort and knowledge of the controller designer are required for implementing all the proposed stages. In that sense, the independent DRL approach is able to learn the whole ball-dribbling behavior autonomously, achieving best performances with respect to the RL-FLC with less human effort and less previous knowledge. An advantage that still remains from the RL-FLC method is the considerably lower RL training time, regarding the DRL scheme (51 episodes vs. 845 episodes approximately for achieving a performance of 30%). In that sense, the transfer knowledge strategies for DRL agents proposed in this work are able to reduce that learning time down up to 225 episodes, opening the door to make achievable future implementations for learning similar behaviors with physical robots.

6.5 Discussion

DRL+NeASh schemes showed either better performances and learning times for the inwalk-kicking problem. By contrast, DRL+NeASh and DRL+CoSh approaches showed similar

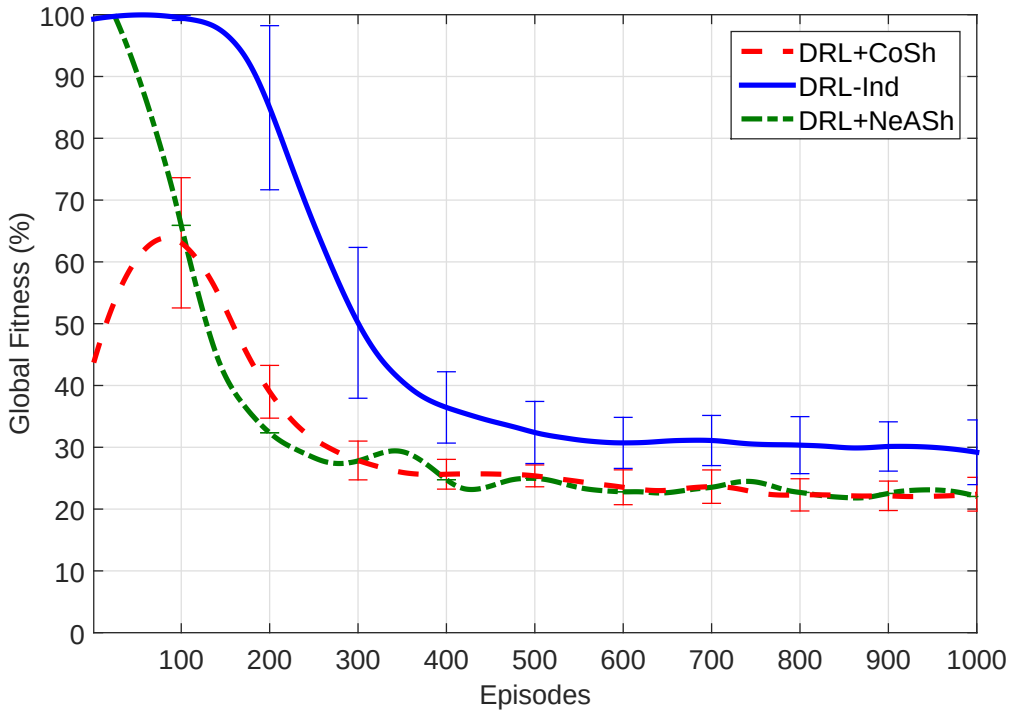

Figure 6.6: Ball-dribbling learning evolution plots for the 3D realistic simulator. Results are averaged across 10 learning runs and error bars show the standard errors.

Approach	**Final performance(%) [Scored Goal Rate]**	**Time to Th.** (30.0%)
DRL+NeASh	22.04	225
DRL-CoSh	22.40	267
DRL-Ind	29.19	845
RL-FLC	34.40	51

Table 6.4: Ball-Dribbling performances (in which lower %s are better).

performances for the dribbling problem. This is an interesting point to be discussed taking into account the quality of the sources of knowledge used by each problem: the inwalk-kicking behavior used a source of knowledge with a performance of around 15% (being 100% the optimal), while the ball-dribbling used a source of knowledge of around 45% (being 0% the optimal). The DRL+NeASh scheme showed the best averaged performance for both problems: 58.53% for the inwalk-kicking, and 22.04% for the ball-dribbling. Note that the source performance for the dribbling case was closer to the optimal policy. Thus, it is possible empirically conclude that benefits of NeASh approach are more noticeable when the source of knowledge has poor performances or more uncertainty; otherwise, the CoSh approach could be more convenient due to its simplicity and easy parameter tuning, and also because of CoSh is able to deal with both, discrete and continuous action spaces.

Video demonstrating the inwalk-kick and ball-dribbling learned policies performed with a real NAO robot can be found online at [37]. The policies are transferred directly to the physical robot, thus, the final performance is dependent on how realistic the simulation platform is.

CoSh and NeASh were able to fulfill all the requirements indicated in Section 6.3: transferring on any RL algorithm that uses an action value function; transferring from different SoK types; including prior-coordination without more complexity than a single-agent RL method; and allowing heterogeneous agents with different source and target tasks but same source and target problem-space. Those considerations were accomplished precisely because of the NeASh and CoSh simplicity. However, since this work is one of the first approaches that address coordination or acceleration of DRL systems, my intention was to introduce basic and simple concepts and methods as a starting point, and as motivation for future researches on this field, in which more sophisticated methods can be used.

6.6 Summary

This chapter presented a DRL architecture to alleviate the effects of the curse of dimensionality and the large number of training trials required to learn tasks in which multi-dimensional action spaces are involved. Three DRL schemes are considered and tested: DRL-Ind, implemented with independent learners and no prior-coordination; DRL+CoSh, accelerated-coordinated by using the Control Sharing (CoSh) knowledge transfer approach, which is extended from the single-agent case to the DRL proposed architecture; and DRL+NeASh, the Nearby Action Sharing (NeASh) KT approach proposed for including a measure of uncertainty to the CoSh procedure.

The presented methods have been validated by implementing two real-world problems, the inwalk-kicking and the ball-dribbling behaviors, both performed with humanoid biped robots, where each component of the requested velocity vector $[v_x, v_y, v_\theta]$ is learned in parallel with independent agents working in a multi-agent task. Results have also shown that even without prior-coordination, both asymptotic convergence and indirect coordination are achieved among DRL-Ind agents. They have shown that it is possible to reduce the training episodes and coordinate the DRL by using knowledge transfer from simple linear controllers, obtaining better performances and learning times with respect to the DRL-Ind scheme.

Chapter 7

Conclusions and Future Direction

This dissertation addressed the Multi-Agent based Decentralized Reinforcement Learning (DRL) of individual behaviors. DRL is proposed as an alternative to alleviate the effects of the curse of dimensionality and the large number of training trials required to learn tasks in which multi-dimensional action spaces are involved. In order to address this, several DRL schemes have been introduced and tested in order to coordinate and accelerate the learning procedure as well as to achieve asymptotic converge. These schemes have been validated through an extensive experimental study in which different problems were successfully described, modeled, and implemented. Results have shown empirically that benefits of MAS are also applicable to complex problems like robotic platforms, by using DRL systems.

Chapter 3 provided an introduction to DRL, its potential advantages and challenges. Then, three DRL algorithms were described: independent DRL scheme (DRL-Ind); DRL-Lenient; and the Cooperative Adaptive Learning Rate (DRL-CA), an author's original contribution. Finally, an overview on related work was provided.

Chapter 4 promoted and proposed a five-stages methodology to model and implement DRL systems in which basic concepts, definitions, and practical implementation issues were presented.

In Chapter 5, the DRL methodology introduced in Chapter 4, as well as the DRL-Ind, DRL-Lenient, and DRL-CA algorithms were validated with an extensive empirical study on four different problems. The experimental results evidenced that DRL implementations show better performances and faster learning times than their CRL (centralized RL) counterparts. 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. The benefits of the proposed MA based methods were more remarkable as the problem complexity increased, in which a CRL scheme is infeasible. 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.

Chapter 6 introduced two Knowledge-Transfer (KT) based DRL schemes: DRL+CoSh, accelerated-coordinated by using Control Sharing; and DRL+NeASh, with Nearby Action

Sharing, which includes a measure of uncertainty to the DRL-CoSh procedure. These two schemes, in addition to DRL-Ind have been validated by implementing two real-world problems: the inwalk-kicking and the ball-dribbling behaviors; both performed with humanoid biped robots. Results have also shown that even without prior-coordination, both asymptotic convergence and indirect coordination are achieved among DRL-Ind agents. They have shown that it is possible to reduce the training episodes and coordinate the DRL by using KT, obtaining faster learning times with respect to the DRL-Ind scheme.

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. Until 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.

Chapter 8

Bibliography

[1] R. A. C. Bianchi, M. F. Martins, C. H. C. Ribeiro, A. H. R. Costa, Heuristically-accelerated multiagent reinforcement learning., IEEE Trans. Cybern. 44 (2) (2014) 252–65.

[2] D. Bloembergen, M. Kaisers, K. Tuyls, Lenient Frequency Adjusted Q-learning, in: 22nd Belgium-Netherlands Conf. Artif. Intel., 2010, pp. 19–26.

[3] 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.

[4] G. Boutsioukis, I. Partalas, I. Vlahavas, Transfer Learning in Multi-Agent Reinforcement Learning Domains, in: H. M. Sanner S. (ed.), Recent Adv. Reinf. Learn. Lect. Notes Comput. Sci. vol 7188., Springer, Berlin, Heidelberg, 2012, pp. 249–260.

[5] M. Bowling, M. Veloso, Multiagent learning using a variable learning rate, Artif. Intell. 136 (2) (2002) 215–250.

[6] M. Brafman, Ronen I. and Tennenholtz, R-max - a general polynomial time algorithm for near-optimal reinforcement learning, J. Mach. Learn. Res. 3 (1) (2003) 213–231.

[7] L. Busoniu, R. Babuska, B. De-Schutter, A comprehensive survey of multiagent reinforcement learning, Syst. Man, Cybern. Part C Appl. Rev. 38 (2) (2008) 156–172.

[8] L. Busoniu, R. Babuska, B. De-Schutter, D. Ernst, Reinforcement learning and dynamic programming using function approximators, CRC Press, Boca Raton, Florida, 2010.

[9] 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.

[10] C. Claus, C. Boutilier, The dynamics of reinforcement learning in cooperative multiagent systems, in: Proc. fifteenth Natl. Conf. Artif. Intell. Appl. Artif. Intell. '98/IAAI '98, Madison, Wisconsin, USA, 1998, pp. 746–752.

[11] R. H. Crites, A. G. Barto, Improving Elevator Performance Using Reinforcement Learning, in: Adv. neural Inf. Process. Syst. (NIPS), vol.8, Denver, USA., 1995, pp. 1017–1023.

[12] 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.

[13] 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.

[14] 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.

[15] F. Fernández, J. García, M. Veloso, Probabilistic policy reuse for inter-task transfer learning, Rob. Auton. Syst. 58 (July 2009) (2010) 866–871.

[16] P. Glorennec, L. Jouffe, Fuzzy Q-learning, in: Proc. 6th Int. Fuzzy Syst. Conf., vol. 2, Barcelona, 1997, pp. 659–662.

[17] 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.

[18] 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.

[19] G. C. How, T. Wu, M. Cutler, J. P., Rapid transfer of controllers between UAVs using learning-based adaptive control, in: Proc. - IEEE Int. Conf. Robot. Autom., Karlsruhe, Germany, 2013, pp. 5409–5416.

[20] K. Hwang, Y. Chen, C. Wu, Fusion of Multiple Behaviors Using Layered Reinforcement Learning, Syst. Man Cybern. - Part A 42 (4) (2012) 999–1004.

[21] A. Kabysh, V. Golovko, A. Lipnickas, Influence Learning for Multi-Agent System Based on Reinforcement Learning, Int. J. Comput. 11 (1) (2012) 39–44.

[22] 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.

[23] W. B. Knox, P. Stone, Combining Manual Feedback with Subsequent MDP Reward Signals for Reinforcement Learning, in: Proc. 9th Int. Conf. Auton. Agents Multi-agent Syst. (AAMAS 2010), International Foundation for Autonomous Agents and Multiagent Systems, Toronto, Canada, 2010, pp. 5–12.

[24] G. Konidaris, I. Scheidwasser, A. Barto, Transfer in reinforcement learning via shared features, J. Mach. Learn. Res. 13 (1) (2012) 1333–1371.

[25] 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.

[26] G. J. Laurent, L. Matignon, N. L. Fort-Piat, The world of independent learners is not markovian, Int. J. Knowledge-based Intell. Eng. Syst. 15 (1) (2011) 55–64.

[27] D. L. Leottau, Decentralized Reinforcement Learning (source code) (2017).
URL `https://github.com/dleottau/Thesis-DRL`

[28] D. L. Leottau, C. Celemin, UCh-Dribbling-Videos (2015).
URL `https://www.youtube.com/watch?v=HP8pRh4ic8w`

[29] D. L. Leottau, C. Celemin, J. Ruiz-del solar, Ball Dribbling for Humanoid Biped Robots: A Reinforcement Learning and Fuzzy Control Approach, in: K. S. Reinaldo A. C. Bianchi, H. Levent Akin, Subramanian Ramamoorthy (ed.), Rob. 2014 Robot World Cup XVIII - Lect. Notes Comput. Sci. 8992, Springer Verlag, Berlin, 2015, pp. 549–561.

[30] D. L. Leottau, K. Lobos-Tsunekawa, F. Jaramillo, J. Ruiz-del Solar, Accelerating Decentralized Reinforcement Learning of Complex Individual Behaviors, Eng. Appl. Artif. Intell. Submitted.

[31] D. L. Leottau, J. Ruiz-del Solar, R. Babuska, Decentralized Reinforcement Learning of Robot Behaviors, Artif. Intell. In Press.

[32] D. L. Leottau, A. Vatsyayan, J. Ruiz-del Solar, R. Babuska, Decentralized Reinforcement Learning Applied to Mobile Robots, in: D. Behnke, S., Sheh, R., Sariel, S., Lee (ed.), Rob. 2016 Robot World Cup XX, Lect. Notes Artif. Intell. 9776, vol. 9776 LNAI, Springer Verlag, Berlin, 2017.

[33] D. L. Leottau, J. M. Yañez, J. Ruiz-del solar, L. L. Forero, J. Ruiz-del solar, Integration of the ROS Framework in Soccer Robotics: the NAO Case, in: V. Behnke, M. Veloso, A. Visser, R. Xiong (eds.), Rob. 2013 Robot World Cup XVII, Lect. Notes Comput. Sci. Vol. 8371, Springer, 2014, pp. 664–671.

[34] D. L. D. 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.

[35] D. L. D. 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.), Rob. Robot Soccer World Cup XIX, Lect. Notes Artif. Intell., vol. 9513, Springer Verlag, Berlin, 2016, pp. 290–302.

[36] C.-K. Lin, A reinforcement learning adaptive fuzzy controller for robots, Fuzzy Sets Syst. 137 (3) (2003) 339–352.

[37] K. Lobos-Tsunekawa, Inwalk-kicking and ball-dribbling videos (2017).
URL `https://www.youtube.com/watch?v=HP8pRh4ic8w`

[38] K. Lobos-Tsunekawa, D. L. Leottau, J. Ruiz-del Solar, Toward Real-Time Decentralized Reinforcement Learning using Finite Support Basis Functions, in: Rob. Symp. 2017, Nagoya, Japan, 2017.

[39] J. Martin, A Reinforcement Learning Environment in Matlab (source code).
URL `https://jamh-web.appspot.com/download.htm`

[40] 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, Francia, 2007, pp. 192–197.

[41] M. J. Mataric, Reward Functions for Accelerated Learning, in: Proc. Elev. Int. Conf. Mach. Learn., Morgan Kaufmann, Boca Raton, Florida, USA, 1994, pp. 181–189.

[42] L. Matignon, G. J. Laurent, N. Le Fort-Piat, Design of semi-decentralized control laws for distributed-air-jet micromanipulators by reinforcement learning, 2009 IEEE/RSJ Int. Conf. Intell. Robot. Syst. (2009) 3277–3283.

[43] K. T. Michael Kaisers, Frequency Adjusted Multi-agent Q-learning, in: 9th Int. Conf. Auton. Agents Multiagent Syst., Toronto, Canada, 2010, pp. 309–315.

[44] L. Panait, S. Luke, Cooperative Multi-Agent Learning: The State of the Art, Auton. Agent. Multi. Agent. Syst. 11 (3) (2005) 387–434.

[45] L. Panait, K. Sullivan, S. Luke, Lenience towards teammates helps in cooperative multi-agent learning, in: Proc. Fifth Int. Jt. Conf. Auton. Agents Multi Agent Syst., Hakodate, Japan, 2006.

[46] L. Panait, K. Tuyls, S. Luke, Theoretical Advantages of Lenient Learners: An Evolutionary Game Theoretic Perspective, J. Mach. Learn. Res. 9 (2008) 423–457.

[47] 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.

[48] 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.

[49] E. Schuitema, Reinforcement Learning on autonomous humanoid robots, Ph.D. thesis, Delft University of Technology (2012).

[50] 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.

[51] 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.

[52] P. Stone, M. Veloso, Multiagent Systems: A Survey from a Machine Learning Perspective, Auton. Robot. 8 (3) (2000) 1–57.

[53] R. Sutton, A. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.

[54] F. W. T. Röfer, T. Laue, J. Müller, A. Fabisch, F. Feldpausch, K. Gillmann, C. Graf, T. J. de Haas, A. Härtl, A. Humann, D. Honsel, P. Kastner, T. Kastner, C. Könemann, B. Markowsky, O. J. L. Riemann, B-human team report and code release 2011, Tech. rep., Department of Computer Science, University of Bremen, Bremen, Germany (2011).

[55] 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.

[56] 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.

[57] A. Taylor, I. Dusparic, V. Cahill, Transfer Learning in Multi-Agent Systems Through Parallel Transfer, in: 30TH Int. Conf. Mach. Learn., Atlanta, USA, 2013, p. 28.

[58] M. Taylor, P. Stone, Transfer learning for reinforcement learning domains: A survey, J. Mach. Learn. Res. 10 (2009) 1633–1685.

[59] 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.

[60] M. E. Taylor, P. Stone, Cross-domain transfer for reinforcement learning, in: Proc. 24th Int. Conf. Mach. Learn. - ICML '07, ACM Press, New York, New York, USA, 2007, pp. 879–886.

[61] 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., vol. 1, 2010, pp. 2397–2403.

[62] 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.

[63] K. Tuyls, P. J. T. Hoen, B. Vanschoenwinkel, An Evolutionary Dynamical Analysis of Multi-Agent Learning in Iterated Games, Auton. Agent. Multi. Agent. Syst. 12 (1) (2005) 115–153.

[64] A. Vatsyayan, Video: centralized and decentralized reinforcement learning of the ball-pushing behavior (2016).
URL `https://youtu.be/pajMkrf7ldY`

[65] 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.

[66] N. Vlassis, A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence, in: R. J. Brachman, T. Dietterich (eds.), Synth. Lect. Artif. Intell. Mach. Learn., No. 2, 1st ed., chap. 2, Morgan and Claypool Publishers, 2007, pp. 1–71.

[67] P. Vrancx, Y. De Hauwere, A. Nowé, Transfer Learning for Multi-agent Coordination, in: 3th Int. Conf. Agents Artif. Intell., Rome, Italy, 2011, pp. 263–272.

[68] C. J. C. H. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (3-4) (1992) 279–292.

[69] 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, vol. 2723 of Lecture Notes in Computer Science, Springer Heidelberg, Berlin, 2003, pp. 356–368.

[70] J. M. Yáñez, P. Cano, M. Mattamala, D. L. Leottau, C. Celemín, P. Saavedra, C. Villegas, K. Lobos, G. Azócar, N. Cruz, R. Pérez, P. Miranda, C. Verdugo, F. Herrera, L. Cossio, J. Ruiz-del-Solar, UChile Robotics Team Team Description for RoboCup 2016, in: Rob. 2016 Robot Soccer World Cup XX Preproceedings, July 2016, Leipzig, Germany, 2016.

[71] 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.

[72] Y. Yujing Hu, Y. Yang Gao, B. Bo An, Accelerating Multiagent Reinforcement Learning by Equilibrium Transfer, IEEE Trans. Cybern. 45 (7) (2015) 1289–1302.

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 comparisons and evaluations are carried out based on the best performance potentially achievable by each method, according to the optimization results. Before each set of optimizations, I try to achieve a good set of parameters by hand-tuning, such as seed, and then it is 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 [5, 7, 44, 59]. However, since only local optima are guaranteed, I 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 A.1, where $paramListV$ is an structure which stores every parameter combination and its respective evaluation value. It is used $neighbours = 4$ and $oneDimPerTry$ enabled for all the experiments in this work. The source code is also shared on-line at Leottau's code repository [27].

Algorithm A.1 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 $n = 1, \cdots, neigbours$ **do**
                $x^{\mathrm{d}} \leftarrow \max(x_0^{\mathrm{d}} - n \cdot paramStep^{\mathrm{d}}, paramMin^{\mathrm{d}})$
                $p \leftarrow GetParamIndex(x)$
                if $ParamListV(p) == \infty$ **then**
                    $ParamListV(p) \leftarrow GetEval(x)$
                end if
            end for
            for neighbor $n = 1, \cdots, neigbours$ **do**
                $x^{\mathrm{d}} \leftarrow \min(x_0^{\mathrm{d}} + n \cdot paramStep^{\mathrm{d}}, paramMax^{\mathrm{d}})$
                $p \leftarrow GetParamIndex(x)$
                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($fval$, $x = GetParam(p)$)