UNIVERSITÀ DEGLI STUDI DI TRIESTE

UNIVERSITÀ DEGLI STUDI DI TRIESTE

XXXIV CICLO DEL DOTTORATO DI RICERCA IN

INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

Reinforcement Learning for Real World Dynamical Systems: Applications and Limitations

Settore scientifico-disciplinare: **ING-INF 04**

DOTTORANDA
ERICA SALVATO

COORDINATORE
PROF. ALBERTO TESSAROLO [illegible]

SUPERVISORE DI TESI
PROF. FELICE ANDREA PELLEGRINO [illegible]

CO-SUPERVISORE DI TESI
PROF. ERIC MEDVET [illegible]

ANNO ACCADEMICO 2020/2021

Reinforcement Learning for Real World Dynamical Systems: Applications and Limitations

PhD Dissertation

by

Erica SALVATO

Master of Science in Control Systems Engineering, University of Trieste, Italy.

Erica Salvato
Reinforcement Learning for Real World Dynamical Systems: Applications and Limitations
PhD Dissertation, February 15th, 2022
Supervisor: Prof. Felice Andrea Pellegrino
Co-Supervisor: Prof. Eric Medvet

University of Trieste
Department of Engineering and Architecture
PhD course in Industrial and Information Engineering
Via Alfonso Valerio 6/1
34100 Trieste

Abstract

Reinforcement learning (RL) offers the chance to make a system learn control policies to successfully and autonomously perform specific tasks. It is often applied to classical and fairly simple artificial domains. Recently, the introduction of new deep learning tools, in combination with the development of more powerful hardware, has made it possible to apply it to real-world tasks as well. Despite the inherent potential of RL as a control technique, however, it still has some limitations that affect its effectiveness on real-world dynamic systems. These limitations become more evident as the complexity of the problems increases. In the present work, we focus on the applications of RL to real-world control problems, both in simulation and in reality. We first present a formalism that will be useful throughout the work. We show two possible applications of RL to the control of real dynamic systems, such as a signalized traffic intersection developed in a simulated environment, and the real Free Electron Laser (FEL) of the FERMI at Elettra Sincrotrone Trieste. Subsequently, we analyze RL as a robotics control tool with a primary focus on the *reality gap* (RG), i.e., the phenomenon, triggered by the difference between simulator and real system, which leads to the degradation of the controller performance, learned on a simulator, when used on the real system. In particular, we are interested in finding a way to characterize and quantify the gap. Therefore, we propose a new index that can grasp and quantify the proneness of a controller to exhibit RG. Finally, we characterize some modeling errors, affecting an open-source simulated robotic platform, that lead to a reduction in controller performance during real-world applications. We use the proposed index to capture the resulting controller performance.

Abstract (Italian Version)

Il Reinforcement Learning (RL) è una branca del Machine Learning (ML) che permette di rendere i sistemi autonomi nell'apprendimento di leggi di controllo per l'esecuzione di specifici task. Sebbene sia spesso studiato su sistemi artificiali e caratterizzati da un comportamento dinamico semplice, recentemente l'introduzione dei nuovi strumenti di deep learning, insieme allo sviluppo di hardware sempre più prestante, ha favorito l'uso dell'RL nel controllo di sistemi dinamici reali. Tuttavia, nonostante il suo potenziale, l'RL presenta ancora alcune limitazioni e non garantisce prestazioni efficaci quando coinvolto nel controllo di tali sistemi. Tali limitazioni diventano più evidenti all'aumentare della complessità dei problemi trattati. Nel presente lavoro, focalizziamo la nostra attenzione sulle applicazioni di RL a problemi di controllo del mondo reale, anche simulati. Per prima cosa presentiamo un formalismo utile per il seguito del lavoro. Mostriamo due possibili applicazioni di RL al controllo di sistemi dinamici reali: un incrocio semaforico, sviluppato in un ambiente simulato, e il Free Electron Laser (FEL) del FERMI di Elettra Sincrotrone Trieste. Successivamente, analizziamo l'RL come tecnica di controllo di robot, concentrandoci in particolare sul problema del *reality gap* (RG): quel fenomeno, causato dalla differenza tra simulatore e sistema reale, che porta al degrado delle prestazioni di un controllore appreso sul simulatore e testato sul sistema reale. In particolare, siamo interessati a trovare un modo per caratterizzare questo gap. Pertanto, proponiamo un nuovo indice in grado di cogliere e quantificare la propensione di un controllore ad esibire un RG. Infine caratterizziamo alcuni errori di modellazione, relativi ad una piattaforma robotica simulata open-source, che portano

a una riduzione delle prestazioni del controllore nel trasferimento da sistema simulato a sistema reale. Applichiamo in tale esempio l’indice proposto per valutare le prestazioni del controllore ottenuto.

Contents

List of Figures

List of Tables

1 Introduction

Reinforcement learning (RL) is a learning model for sequential decision-making tasks. Along with supervised learning and unsupervised learning, it is one of the pillar paradigms of machine learning (ML).

The main goal of RL is to find a sequence of control inputs that can guide a dynamical system to maximize a reward assuming minimal, or even null, knowledge about the system response to those inputs. This result is achieved by performing *trial and error* interactions with the system to be controlled. Specifically, similar to the control theory, RL studies how to use past experience to improve the future behavior of a dynamic system [142]. Although it has recently proven its effectiveness as a control tool for many toy artificial examples [113, 156, 98], RL is not yet able to fully express its potential in real-world, physics-based control systems [50].

In the present work, we refer to *real-world* systems as dynamical systems involved in real-world control problems (e.g., factory chains, vehicles, robots, etc.) irrespective of being simulated or not.

Real-world tasks are mainly characterized by nonlinear dynamics, often related to different, interconnected and mutually dependent processes, influenced by random factors that are difficult to predict. Moreover, they might be characterized by partially observable states, which limit access to the system behavior. Not least, they could also be subject to latencies, which pose the challenge of dealing with significant delays in the effect of control inputs on system dynamics. All the above difficulties differ in severity depending on the problem faced, and are not easy to determine a priori. However, they arise for any type of control technique, and how to deal with them is still an open challenge for applying control theory tools in real life.

In RL, where learning is data-driven and not based on mathematical modeling of system dynamics, one further challenge is to find solutions that are data-efficient, able to work with limited exploration (dangerous and expensive to be performed on a real facility), and able to be aware of safety constraints without the involvement of external watchdogs. Therefore, leveraging effective RL approaches on real-world systems is a quite tricky challenge.

There are several attempts in the literature to address these challenges individually. For example, in [113, 74, 174] the sample efficiency problem, i.e., the ability to learn in a data-limited domain thus finding a data-efficient solution, is tackled by using expert demonstration for agent bootstrapping. Other solutions propose to learn dynamical model ensembles on which different sampling strategies can be used in order to properly drive the exploration [38, 30]. Recent works [43, 4, 149] face the problem of safety operational constraints by defining the decision making strategy as constrained Markov decision processes (CMDPs). In [32], alternatively, authors propose a meta-gradient approach to properly balance the maximization of the reward and constraints violation minimization.

Another promising approach is the use of simulators for the training phase. The goal is to learn a controller on a digital twin of the plant, and then reuse it on the real one, thus speeding up the training and avoiding dangerous action selections. However, this approach is particularly dependent on the simulator ability to simulate the behavior of the real system, and it is often compromised by the phenomenon of the *reality gap*, i.e., the deterioration of the controller performance in the real world due to the plant-model mismatch [148]. In the worst case, the controller transfer from simulator to real system turns out to be ineffective even if it works perfectly on the simulator.

Based on the above considerations, the following are the research questions motivating the studies conducted in this work:

(Q1): Given the potential of RL as a control technique on simple dynamical systems, can it be used to control dynamical systems that address real-world practical problems?

(Q2): Is there a way to quantify the reality gap? Can we use this measure to characterize a learning technique in a deeper way than just considering effectiveness and efficiency?

With the aim of answering these research questions, we focus on the application of RL to real-world control problems, both in simulation and in reality. We first introduce a control systems-oriented formal framework of the RL. We show two effective applications of RL to the control of real dynamic systems, such as a signalized traffic intersection developed in a MATLAB simulated environment, and the real FEL facility at the FERMI of Elettra Sincrotrone Trieste. Subsequently, we analyze the RL in robotics and the *reality gap* (RG). We survey relevant and recent literature concerning RG in the context of RL and outline the three main approaches for coping with RG. We propose an abstract formulation of the RG problem that applies to the general scenario where a learning algorithm is used for inferring a policy: the formulation applies to RL as well to other kinds of techniques, such as Evolutionary Computation or Model Predictive Control, and different domains. As a key component of this formulation, we propose a novel index (called L,ϕ-gap and detailed in later sections) that measures the proneness of a learning algorithm to exhibit the RG. L,ϕ-gap can be used along with common indexes measuring effectiveness and efficiency for assessing and comparing learning algorithms. Finally, we characterize some modeling errors, affecting an open-source simulated robotic platform, that lead to a reduction in controller performance during real-world applications.

We use the L,ϕ-gap to quantify the reality gap in a sim-to-real transfer of the resulting controller.

2 Reinforcement Learning for control systems

Reinforcement Learning (RL) [158] allows to design controllers (often referred to as *agents*) with the capability of learning an optimal behavior by interacting with the *environment*. The behavior is defined in terms of state-action pairs, also known as *policy*, which is learned through a *trial and error* process.

In practice, it can be employed to perform optimal data-driven control without the need to rely on a mathematical model of system dynamics [158, 19, 31].

It is typically described as a Markov Decision Process (MDP); i.e., a tuple (X, A, p, h) defined by the state set X, the control set A, the transition probability from state $x \in X$ to state $x' \in X$ in 1 steps $p := P\{x^{(k+1)} = x',\ x^{(k)} = x\}$, and the reward function $h : X \times A \to \mathbb{R}$. In brief, it expresses a discrete-time stochastic control process in which the reward function is employed to assess the quality of the controller choice in terms of task achievement.

Here, we adopt a control systems-oriented formalism similar to the one employed in [22]. However, what follows also applies to MDP settings by defining some of the following elements in terms of expected values.

2.1 Adopted formalism

A *dynamical discrete-time system* Ω is a tuple (X, A, O, f, g) composed of the state set X, the control set A, the observation set O, a transition function $f : X \times A \to X$, and an observation function $g : X \to O$. Let $x^{(k)} \in X$, $a^{(k)} \in A$, and $o^{(k)} \in O$ be the state, the applied control input, and the observation, respectively, at the k-th time-instant. A dynamical discrete-time system, starting from an initial state $x^{(0)}$ and subjected to a control sequence $a^{(0)}, a^{(1)}, \dots,$ evolves according to the following laws:

$$x^{(k+1)} = f\left(x^{(k)}, a^{(k)}\right) \tag{2.1}$$

$$o^{(k+1)} = g\left(x^{(k+1)}\right) = g\left(f\left(x^{(k)}, a^{(k)}\right)\right). \tag{2.2}$$

An *environment* E is a dynamical discrete-time system described by a tuple (X, A, O, f, g, h) composed of the same elements of Ω plus a *reward function* $h : X \times A \to \mathbb{R}$. Let $r^{(k+1)} \in \mathbb{R}$ be the reward at the $(k+1)$-th time-instant. An environment E, starting from an initial state $x^{(0)}$ and subjected to a control sequence $a^{(0)}, a^{(1)}, \dots,$ evolves according to Equations (2.1) and (2.2) and:

$$r^{(k+1)} = h\left(x^{(k)}, a^{(k)}\right). \tag{2.3}$$

Overall, we assume that $g(f(\cdot)) = f(\cdot)$, thus resulting in $o^{(k+1)} = x^{(k+1)}$. The following is also pertinent, with appropriate adjustments, in case of $g(f(\cdot)) \neq f(\cdot)$ that simply means dealing with a partial observability of the state. In the MDP setting, this amounts to employing a Partially Observable Markov Decision Process (POMDP), which requires to introduce the concept of *belief* [143]. However, it is not necessary for the purpose of the present work.

A controller (or *policy*) for a dynamical discrete-time system, and therefore also for an environment, is a function $\pi : O \to A$.

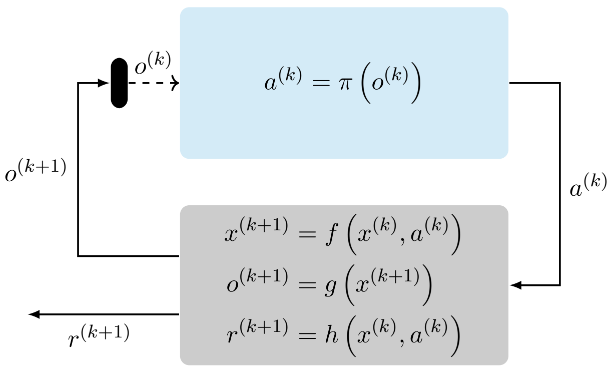

$o^{(k)}$
$a^{(k)} = \pi\left(o^{(k)}\right)$
$o^{(k+1)}$
$a^{(k)}$
$x^{(k+1)} = f\left(x^{(k)}, a^{(k)}\right)$
$o^{(k+1)} = g\left(x^{(k+1)}\right)$
$r^{(k+1)} = h\left(x^{(k)}, a^{(k)}\right)$
$r^{(k+1)}$

Figure 2.1: The closed-loop system $\pi \leftrightarrow E$, where a policy (light blue block above) applies a control input $a^{(k)}$ to an environment (gray block below) that outputs an observation $o^{(k+1)}$ and a reward $r^{(k+1)}$.

Given a discount factor $\gamma \in [0,1]$, an *optimal policy* π^* is a policy that satisfies, for any initial state $x^{(0)}$:

$$\begin{aligned}\pi^* &= \arg\max_{\pi\in\Pi} \sum_{k=0}^{+\infty} \gamma^k h\left(x^{(k)}, \pi\left(o^{(k)}\right)\right) \\ &= \arg\max_{\pi\in\Pi} J_\pi\left(x^{(0)}\right), \end{aligned} \tag{2.4}$$

where Π is the set of policies, while $J_\pi\left(x^{(0)}\right)$ is the *infinite horizon discounted reward* starting from $x^{(0)}$ under the policy π.

We denote by $\pi \leftrightarrow E$ the closed-loop system where π determines the control input to be applied on E, represented in Figure 2.1. In particular, according to the above definitions and to the assumption that $g(f(\cdot)) = f(\cdot)$, the scheme represents a *state feedback* control. Clearly, when $g(f(\cdot)) \neq f(\cdot)$, the same scheme could represent an *output feedback* control.

A *policy learning algorithm* L is an algorithm that, given an environment $E = (X, A, O, f, g, h)$ and a *learnable policies set* $\Pi^L \subseteq \Pi$, outputs (learns) a policy $\pi^L = L(E) \in \Pi^L$. When the learning encompasses an interaction with the environment, the policy learning algorithm can be seen as an *agent* L that learns a controller π by interacting with an environment E. We denote by $L \leftrightarrow E$ the resulting closed-loop system

(Figure 2.2). A policy learning algorithm is said to be *model-based*, if it relies on the knowledge of f and g (either known in advance, or identified based on collected data), or *model-free*, otherwise.

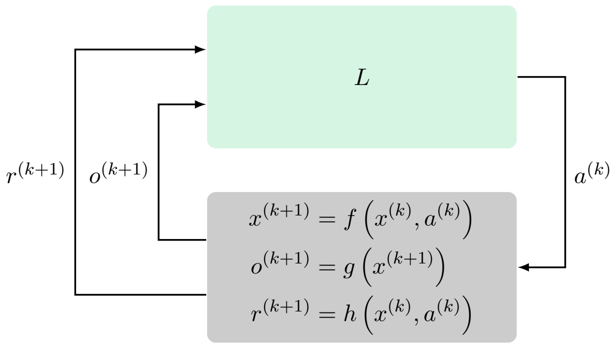

L
$r^{(k+1)}$
$o^{(k+1)}$
$a^{(k)}$
$x^{(k+1)} = f\left(x^{(k)}, a^{(k)}\right)$
$o^{(k+1)} = g\left(x^{(k+1)}\right)$
$r^{(k+1)} = h\left(x^{(k)}, a^{(k)}\right)$

Figure 2.2: The closed-loop system $L \leftrightarrow E$ where a RL-based agent L (green block above) applies a control input $a^{(k)}$ to an environment (gray block below) that outputs an observation $o^{(k+1)}$ and a reward $r^{(k+1)}$.

A typical RL agent can be seen as a policy learning algorithm L. Indeed, during training, it interacts with the environment $E = (X, A, O, f, g, h)$ and updates a controller π by observing the consequences (in terms of reward r) of the selected control inputs. Its global goal is to learn a controller π^* in E which satisfies (2.4).

The learning procedure can either be *episodic* or continuous (*non-episodic*): in the former case, the learning is performed through episodes and the state is reset in case of a failure, a goal achievement, or the achievement of the maximum episode length T. In the latter, the learning proceeds without interruptions. The training typically ends when the discounted reward settles in average on a constant value (converges). In addition, the agent can update the controller either by using data from the current policy (*on-policy*) or independently of it (*off-policy*).

RL approaches can be categorized in three main categories [158]:

- *Value-function* approaches, based on the idea of the *value* of a state (value function $V_\pi(x)$) or of a state-control input pair (action-value function $Q_\pi(x,a)$). The value function and the action-value function represent, respectively, the cumulative reward obtained from x by applying π and the cumulative reward obtained from x by first choosing an a and then applying π. The optimal policy corresponds to the optimal V or Q functions:

$$V^*(x^{(k)}) = \max_\pi J_\pi \left(x^{(k)}\right) \tag{2.5}$$

$$Q^*(x^{(k)}, a^{(k)}) = \max_\pi h\left(x^{(k)}, a^{(k)}\right) + \gamma J_\pi \left(x^{(k+1)}\right). \tag{2.6}$$

In brief, by interacting with the environment and observing rewards, either V^* or Q^* are estimated, thus leading respectively to the optimal policies:

$$\pi^*(x^{(k)}) = \arg\max_{a \in A} [h(x^{(k)}, a) + V(f(x^{(k)}, a))] \tag{2.7}$$

$$\pi^*(x^{(k)}) = \arg\max_{a \in A} Q\left(x^{(k)}, a\right) \tag{2.8}$$

- *Policy-search* approaches, in which a parametrized policy π_θ is defined, whose parameters θ are updated based on the observed reward in order to maximize J_{π_θ}, by employing either gradient-based or gradient-free optimization techniques [171].

- *Actor-Critic* approaches, which integrate the idea of both previous categories. $V_\pi(x)$, in this case, is employed as a baseline (Critic) for policy gradient optimization (Actor).

Further technicalities are not necessary for the purpose of this work. For additional details on RL we refer readers to [158, 46, 22, 142].

3 RL applications for real world dynamical system

Although RL is often validated on classical and rather simple examples, significant progress has recently been made to apply it to real world problems.

In the following, we provide two different case-studies of RL controlled real world dynamical systems.

In Section 3.1 we present a simulated signalized urban intersection where autonomous vehicles (AVs) and human-driven vehicles (HDVs) coexist. Here, the RL agent plays the role of the traffic-lights controller, and we use a non-episodic tabular action-delayed Q-Learning as learning algorithm.

In Section 3.2 we present the second case study concerning the free-electron laser facility (FEL) at the FERMI of Elettra Sincrotrone Trieste. The RL experiments are performed directly on the real facility and do not involve any simulator. We apply two different learning algorithms in order to face two different control problems: (a) an episodic Q-learning with linear function approximation, to make the agent able to find an optimal working point, starting from random initial conditions; and (b) a non-episodic Natural Policy Gradient REINFORCE algorithm, to recover the optimal working point when some drifts, or working conditions changes, occur.

Both the proposed applications lead us to highlight the potential of the RL paradigm in solving real control problems.

3.1 Control of a mixed autonomy signalized urban intersection

We consider a mixed autonomy traffic intersection where the traffic intersection controller decides whether the traffic-lights will be green or red at each lane for multiple traffic-light blocks (TLBs). The objective of the traffic intersection controller is to minimize the queue length at each lane, thus maximizing the outflow of vehicles over each TLB.

We assume that the traffic intersection controller informs autonomous vehicles (AVs) whether the traffic-light will be green or red for the future traffic-light block. Thus, the AVs can adapt their dynamics by solving individual optimal control problems. Human-driven vehicles (HDVs), on the other hand, can only observe the current state of the traffic-light, thus adapting their cruise dynamics. We model the decision process of the traffic intersection controller E_1 as a discrete-time deterministic delayed MDP (DDMDP). In particular, we propose two different discretizations for the DDMDP: (1) a fixed timing approach, in which each TLB has a predefined fixed duration, and (2) an event driven approach, in which the decision making is triggered when some predefined events occur, thus resulting in a variable TLB duration. We use a tabular Q-Learning algorithm L_1 in a non-episodic framework to obtain the optimal policy π_1^*.

We numerically investigate the proposed approach as the AVs proportion inside the intersection increases.

3.1.1 Motivations

AVs have the potential to disrupt traffic-intersection control technologies. In a hypothetical AVs-only scenario, the traffic-intersection controller may plan vehicles motions to avoid collisions at the intersection. However, in a more realistic scenario, HDVs and AVs will co-exist, thus giving rise to *mixed-autonomy* intersections. In this setting, the traffic intersection controller cannot communicate with the HDVs to influence their dynamics, and traffic-lights are necessary to avoid collisions.

Against this backdrop, the research challenge becomes the one of designing the traffic-intersection controller for *mixed autonomy* by exploiting the presence of AVs.

Finding an optimal solution to the problem at hand is computationally challenging. The overall dynamics of the intersection, and consequently the current traffic condition, (1) depends on the dynamics of each vehicle, which in turn is influenced by the dynamics of all neighboring vehicles, and (2) turns out to be unpredictable due to the presence of HDVs. In addition, the traffic-intersection controller decision affects the dynamics of the vehicles in a non-linear and non-smooth manner.

3.1.2 Related literature

Algorithms to control the traffic-light duration at the urban intersection based on dynamic programming or on control theory have been proposed, for instance, in [163, 60, 34, 124, 102]. For a detailed literature overview, we refer the reader to [132, 58]. Solutions using RL to cope with complex optimization problems and uncertainties have been proposed in [181, 182, 2, 37]. See also [13] for a broad review. However, all the above works did not consider the scenario where AVs can coexist with HDVs. Recently, [178, 177] considered RL-based algorithms for mixed autonomy. However, all the above papers did not consider the possibility that the AVs can be informed about when they can enter the intersection. Some authors considered traffic-light-free intersection control designs when there are only AVs [189]. Several other authors proposed decentralized algorithms based on the coordination among the AVs [48, 80]. However, soon AVs and HDVs will coexist. Hence,

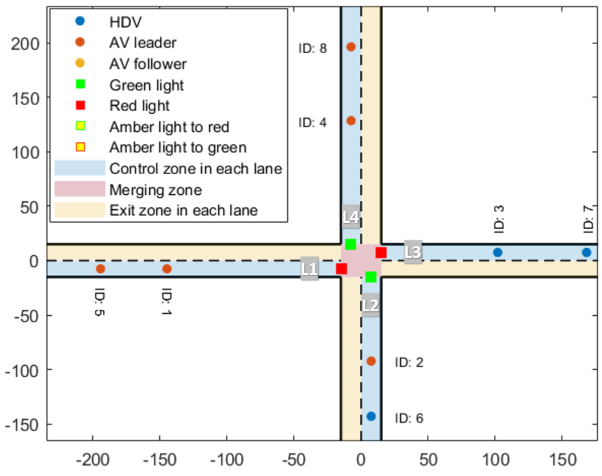


Figure 3.1: Traffic-light controlled intersection in a mixed autonomy scenario

those approaches can not be applied when HDVs are present, since the traffic-light will control AVs movements only.

3.1.3 The urban intersection system

We consider a signalized urban intersection consisting of 4 lanes (Figure 3.1). We partition the urban intersection in three parts: (1) a *Merging Zone* (MZ) of size $L_M \times L_M$, delimiting the area where vehicles of different lanes converge; (2) a *Control Zone* (CZ) of length L_C for each lane, where vehicles travel before entering the MZ; (3) an *Exiting Zone* (EZ) of length L_E for each lane, where vehicles travel after crossing the MZ.

A vehicle is considered to exit the intersection when it covers a distance of $L_C + L_M + L_E$. A traffic-light is placed at the junction between the CZ and the MZ of each lane (4 traffic-lights in total). Each vehicle can enter the MZ when the respective traffic-light is in a green status and the vehicle approaches L_C. Conversely, a vehicle stops within the CZ when the respective traffic-light exhibits a red status and the vehicle is close to L_C. Once the vehicles enter the MZ they cross the intersection and do not stop.

We now introduce some notations which we use throughout this chapter. We denote by $c(i, j)$ the i-th vehicle at lane j, where $i \in \{1, \ldots, N^j_{\text{max}}\}$, with N^j_{max} maximum number of vehicles admitted in the intersection at the j-th lane, and $j \in \{1, 2, \ldots, n_l\}$, with $n_l = 4$ (cf. Figure 3.1). A new vehicle $c(i', j)$ entering the CZ of the j-th lane right after $c(i, j)$ will have $i' = i + 1$, i.e., the more recent the vehicle access to the intersection, the higher the index i associated with it will be.

v_{max} is the maximum allowable speed within the intersection system. A vehicle $c(i, j)$, entering the CZ of the j-th lane at time $t = t^0_{i,j}$ and traveling with a constant speed v_{max} is supposed to enter the MZ at $t^m_{i,j} = t^0_{i,j} + \frac{L_C}{v_{\text{max}}}$. The above represents the time at which the vehicle $c(i, j)$ would enter the merging zone without any traffic. We denote by $C^{(t_k)}$ the set of N vehicles in all the lanes of the intersection system at $t = t_k$. We partition $C^{(t_k)}$ in two subsets $C^{(t_k)}_{HD}$ and $C^{(t_k)}_{A}$, respectively denoting the HDVs and the AVs in all the lanes of the intersection system at $t = t_k$. We assume that $C^{(t_k)}_{A} \cap C^{(t_k)}_{HD} = \emptyset$.

We denote by $p_{i,j}(t_k)$, $v_{i,j}(t_k)$, and $u_{i,j}(t_k)$ respectively the position, the speed, and the acceleration of the vehicle $c(i, j)$ in the intersection at $t = t_k$. Each i-th vehicle entering the control zone of the j-th lane at $t = t_i$ will be initialized with an initial position $p_{i,j}(t_i) = 0$. We assume as positive travel direction the one toward the EZ.

Definition 1. Given two vehicles of the same j-th lane, $c(k, j)$ and $c(i, j)$, if $k = i - 1$, then $c(k, j)$ is the *front vehicle* of $c(i, j)$, i.e., the immediately preceding vehicle of $c(i, j)$.

Definition 2. A pair of lanes (j, k) are *non-conflicting* if there are no intersection points that can lead to vehicles crashes. Let $\mathcal{L}$ be the set of non-conflicting pairs.

With reference to Figure 3.1, lanes 1 and 3 are non-conflicting. However, lanes 2 and 1 are conflicting ($\mathcal{L} = \{(1, 3), (2, 4)\}$). A traffic-controller can only make traffic-lights green simultaneously for the non-conflicting pair of lanes.

We assume the following:

Assumption 1. *A vehicle $c(i,j) \in C$ can only move forward or stay still; i.e., no turning, backward gears, or lane changing are allowed.*

Assumption 2. *A vehicle $c(i,j) \in C_A$ is considered sensors equipped and connected to the other autonomous vehicles. $c(i,j)$ is able to estimate $p_{i-1,j}(t)$ and $v_{i-1,j}(t)$ if $c(i-1,j) \in C_{HD}$, while can access the actual values of $p_{i-1,j}(t)$ and $v_{i-1,j}(t)$ if $c(i-1,j) \in C_A$.*

The second assumption entails that an AV can adapt its dynamics to that of the preceding vehicle.

3.1.4 The fixed RL timing approach

In this section, we introduce the first proposed approach for traffic intersection managing in the mixed autonomy urban intersection of Section 3.1.3.

We assume each traffic-light block (TLB) of fixed time duration T_{RL}. The traffic intersection controller decides whether the traffic-light will be green or red at a lane for each TLB. An amber light of fixed duration T_{alert} is included at each stage of traffic-light switching, i.e., a TLB preceeded by a traffic-light switch has a red or green duration of $T_{\text{RL}} - T_{\text{alert}}$. Note that consecutive TLBs of red or green lights are allowed. Thus, if, for example, for m-consecutive TLBs the traffic-light controller selects the red status, then the traffic-light is red for $mT_{\text{RL}} - T_{\text{alert}}$ duration.

At each k-th TLB, the traffic intersection controller decides traffic-light for the $k + d_a$-th TLB, i.e., the decision of the traffic intersection controller at t_k is implemented at $t_k + T_{\text{delay}}$, where $T_{\text{delay}} = d_a\, T_{\text{RL}}$. The delay is due to two reasons: (i) the traffic intersection controller needs time to compute the optimal decision; (ii) the AVs able to optimize their own dynamics need to be informed by the traffic intersection controller of the exact TLB at which they can enter the intersection. The current traffic signal is therefore the result of a past control input of the intersection controller.

Assumption 3. *A vehicle $c(i,j) \in C_A$ optimizes its dynamics only if it is the first vehicle of the lane or if its* front *vehicle is d_{follow} distant from $c(i,j)$, i.e., $p_{i-1,j}(t) - p_{i,j}(t) \geq d_{follow}$.*

Note that the HDVs follow the traffic-lights only.

The goal of the RL-based traffic-light controller is to: minimize the queue length in each lane, thus maximizing the rate of vehicles outflow.

In the following we illustrate the HDVs and AVs dynamics on the basis of the traffic intersection controller decision (Section 3.1.4.1 and Section 3.1.4.2). In Section 3.1.4.3 we describe the decision making strategy of the RL agent and, finally, the experimental results obtained with the proposed approach (Section 3.1.4.5).

3.1.4.1 Human driven vehicle

The dynamics of a HDV is described using the Intelligent Driver Model (IDM) [167]. It is an easy-to-tune adaptive cruise control system able to avoid vehicles collision in car-following mode. The dynamics for a general $c(i,j) \in C_{HD}$ having a front vehicle $c(i-1,j) \in C$ is defined by:

$$\dot{v}_{i,j}(t) = u_{\max} \left(1 - \left(\frac{v_{i,j}(t)}{\bar{v}_{i,j}} \right)^4 - \frac{\left(s^*_{\{i,i-1\}}(t) \right)^2}{s^2_{\{i,i-1\}}(t) + \xi^2} \right), \qquad (3.1)$$

where $\bar{v}_{i,j}$ is the desired speed of the (i,j)-th vehicle, u_{max} is the maximum acceleration, $s_{\{i,i-1\}}(t) = p_{i-1,j}(t) - p_{i,j}(t)$ is the current inter-vehicle distance, $\xi \in \mathbb{R}$ a small positive constant value, and $s^*_{\{i,i-1\}}(t)$ the desired inter-vehicle distance:

$$s^*(v_{i,j}(t),\ \Delta v_{\{i,i-1\}}(t)) = s^0_{i,j} + T_{i,j}v_{i,j}(t) + \\ + \frac{v_{i,j}(t)\ \Delta v_{\{i,i-1\}}(t)}{2\sqrt{u_{\text{max}}\ u_{\text{min}}}}, \tag{3.2}$$

where $\Delta v_{\{i,i-1\}}(t) = v_{i-1,j}(t) - v_{i,j}(t)$ is the difference in speed of two subsequent vehicles, u_{min} is the maximum deceleration, $s^0_{i,j}$ is the jam distance (the minimum desired distance in a traffic jam), and $T_{i,j}$ the safety time gap between two vehicles.

When the vehicle $c(i,j)$ observes a red-light and the vehicle $c(i-1,j)$ passes the CZ, then the vehicle $c(i,j)$ needs to decelerate irrespective of the dynamics of the vehicle $c(i-1,j)$. Here, we consider the red-light as a static vehicle situated at the end of the CZ of the lane. Thus, $\Delta v_{\{i,\text{TL}\}}(t) = -v_{i,j}(t)$, while $s_{\{i,\text{TL}\}}(t) = p_{\text{TL}}(t) - p_{i,j}(t)$ is the current distance of the vehicle $c(i,j)$ from the traffic-light where $p_{\text{TL}}(t) = L_C$ and $v_{\text{TL}}(t) = 0$ for all t. The last term in the right-hand side of Equation (3.1) is replaced by the following term:

$$\frac{\left(s^*_{\{i,\text{TL}\}}\right)^2}{s^2_{\{i,\text{TL}\}}(t) + \xi^2} \tag{3.3}$$

where $s^*_{\{i,\text{TL}\}}$ is the desired distance from the traffic-light replacing $\Delta v_{\{i,i-1\}}(t)$ with $\Delta v_{\{i,\text{TL}\}}(t) = -v_{i,j}(t)$.

When the vehicle $c(i,j)$ is d_{follow} distance away from the preceding vehicle, or there is no other vehicle in the lane, then the vehicle $c(i,j)$ is not sensitive to the dynamics of the preceding vehicle. An HDV $c(i,j)$ having an inter-vehicle distance $s_{\{i,i-1\}}(t) > d_{\text{follow}}$ and observing a green light follows Equation (3.1) with $s_{\{i,i-1\}}(t) = \infty$. On the other hand, the vehicle $c(i,j)$ facing the red-light will follow Equation (3.1) with $s_{\{i,i-1\}}(t) = \infty$ till the distance from the MZ becomes less than d_{follow}. Since the red-light is considered to be a static vehicle at the position $p_{\text{TL}}(t) = L_c$, the dynamics of the vehicle $c(i,j)$ will be the same as the Equation (3.1), where the last term is replaced by the expression (3.3).

We are only left to describe the dynamics when the vehicle $c(i,j)$ has no front vehicles in the CZ and faces the amber light. If $p_{i,j}(t) - p_{\text{TL}}(t) \leq d_{\text{follow}}$, then the vehicle $c(i,j)$ will follow Equation (3.1) with the preceding vehicle either at ∞ or at $p_{i,j}(t)$ if $p_{i-1,j}(t) - p_{i,j}(t) \leq d_{\text{follow}}$. If $p_{i,j}(t) - p_{\text{TL}}(t) > d_{\text{follow}}$, then the vehicle $c(i,j)$ will behave as if the traffic-light was red. Intuitively, if the amber-light is switched on and the vehicle $c(i,j)$ is very close to the intersection, it will follow its dynamics, otherwise, the vehicle would decelerate to stop.

3.1.4.2 Autonomous vehicle

The AV $c(i,j)$, far away from its front vehicle $c(i-1,j)$ (i.e., $p_{i-1,j}(t) - p_{i,j}(t) \geq d_{\text{follow}}$), or that has no front vehicles, seeks to optimize its acceleration/deceleration profile $u_{i,j}(t)$.

In general, if at $t = t_k$ the $c(i,j)$ AV with $p_{i,j}(t_k) = p_{t_k}$ and $v_{i,j}(t_k) = v_{t_k}$, is informed by the intersection controller that at $t = t_k + T_{\text{delay}}$ the traffic-light will be set to a green status, then $u_{i,j}(t)$ will be the solution of the following optimization problem:

$$\begin{aligned}
&\mathcal{P}_1 : \min_{u_{i,j}(\cdot)} \frac{1}{2}\int_{t_k}^{t_{i,j}^m} u_{i,j}^2(t)dt,\\
\text{subject to: } & \dot{v}_{i,j}(t) = u_{i,j}(t),\ \dot{p}_{i,j}(t) = v_{i,j}(t),\\
& p_{i,j}(t_k) = p_{t_k},\ v_{i,j}(t_k) = v_{t_k},\\
& v_{i,j}(t) \leq v_{\max},\ v_{i,j}(t) \geq 0\, \forall\, t \in \left[t_k,\ t_{i,j}^m\right],\\
& p_{i,j}(t_{i,j}^m) = L_C,\\
& t_{i,j}^m \geq t_k + T_{\text{delay}} + T_{\text{alert}}\\
& t_{i,j}^m \leq t_k + T_{\text{delay}} + T_{\text{alert}} + T_{\text{RL}}.
\end{aligned} \tag{3.4}$$

The AV tries to minimize the total energy cost which is represented as the integral of the square of $u_{i,j}(t)$. The first constraint represents the initial conditions and the vehicle dynamics. $t^m_{i,j}$ is the time at which the AV will enter the MZ, thus, the position at that time must be L_C. The velocity must be constrained between 0 and v_{max}. The optimization binds the vehicle to not enter the MZ before the start of the $k+d_a$-th TLB, hence $t^m_{i,j} \geq t_k + T_{\text{delay}} + T_{\text{alert}}$. Note that T_{alert} is 0, if the decision of the traffic intersection controller is the same for the $k-1$-th and the k-th TLBs. The decision variables are $u_{i,j}(\cdot)$ and $t^m_{i,j}$.

We relax the constraints and add the following penalties in the objective to the cost in the problem (3.4):

$$\begin{aligned}
&K_{v_{\text{max}}}\{\max(0,\ (v_{i,j}(t)\ -\ v_{\text{max}}))\},\\
&K_{v_{\text{min}}}\{\max(0,\ -v_{i,j}(t))\},\\
&K^1_{t^m_{i,j}}\{\max(0,(t_k\ +\ T_{\text{delay}}\ +\ T_{\text{alert}})\ -\ t^m_{i,j})\},\\
&K^2_{t^m_{i,j}}\{\max(0,t^m_{i,j}\ -\ (t_k\ +\ T_{\text{delay}}\ +\ T_{\text{alert}})\ +\ T_{\text{RL}}\}.
\end{aligned}$$

For $t \geq t^m_{i,j}$ we set a constant maximum speed profile ($v_{i,j}(t) = v_{\text{max}}$), hence, the AV will move at the highest possible speed at the merging zone which increases the throughput. The delay d_a must be chosen such that there always exists a solution of the optimization problem.

In contrast, if at $t = t_k$ the AV (i,j) is informed by the intersection controller that at $t = t_k + T_{\text{delay}}$ the traffic-light will be set to a red status, then the deceleration will be the solution of the following optimization:

$$\begin{aligned}
&\mathcal{P}_2 : \min_{u_{i,j}(\cdot)} \ \frac{1}{2}\int_{t_k}^{t_{\text{stop}}} u^2_{i,j}(t)dt,\\
\text{subject to: } &\dot{v}_{i,j}(t) = u_{i,j}(t),\ \dot{p}_{i,j}(t) = v_{i,j}(t),\\
&p_{i,j}(t_k) = p_{t_k},\ v_{i,j}(t_k) = v_{t_k},\\
&v_{i,j}(t) \leq v_{\text{max}},\ v_{i,j}(t) \geq 0\ \forall\ t\ \in\ [t_k, t_{\text{stop}}],\\
&p_{i,j}(t_{\text{stop}}) = L_C - \delta, v_{i,j}(t_{\text{stop}}) = 0.
\end{aligned} \tag{3.5}$$

The decision variables are $u_{i,j}(\cdot)$ and t_{stop}. Also in this case we relax the optimization problem by relying on a penalty function approach:

$$\begin{aligned}&K_{v_{\text{max}}}\{\max(0,\ (v_{i,j}(t)\ -\ v_{\text{max}}))\},\\&K_{v_{\text{min}}}\{\max(0,\ -v_{i,j}(t))\}.\end{aligned}$$

Compared to $\mathcal{P}_1$, in $\mathcal{P}_2$ the AV (i,j) stops at time t_{stop} at a position δ away from the MZ. The constraint $p_{i,j}(t_{\text{stop}}) = L_C - \delta$ ensures that when the traffic-light becomes green, the AV can accelerate and enter the intersection at the maximum speed. The AV $c(i,j)$ will update its dynamics, either solving the relaxed version of $\mathcal{P}_1$ or of $\mathcal{P}_2$, each time that a different future traffic-light is selected by the traffic-intersection controller. Note that if the decision of the traffic intersection controller at the $k+1$-th TLB is equal to the one of the k-th TLB, the AV $c(i,j)$ will not updates its dynamics.

However, all the above definitions hold either (1) when an AV enters the intersection at $t = t_k$, initial instant of the k-th TLB, or (2) during the AVs cruise. If the AV $c(i,j)$ enters the intersection at $t\ =\ t_l\ =\ t_k\ +\ T_{\text{RL}}\ -\ \tau$, with $0 < \tau \leq T_{\text{RL}}$, depending on the traffic-light chosen at t_k, the AV will solve one among $\mathcal{P}_1$ and $\mathcal{P}_2$, where the lower time bound of the integration is set to t_l.

Given the condition $t^m_{i,j} \leq t_k\ +\ (d_a+1)T_{\text{RL}}\ +\ T_{\text{alert}}$, the feasibility of $\mathcal{P}_1$ depends on τ. Indeed, in the worst case, the $c(i,j)$ AV entering the CZ at t_l should be able to cover at least the L_C distance at the maximum allowed speed v_{max} within a time interval $[t_l, t_k\ +\ (d_a + 1)T_{\text{RL}}\ +\ T_{\text{alert}}]$:

$$\frac{L_C}{t_k\ +\ (d_a+1)T_{\text{RL}}\ +\ T_{\text{alert}}\ -\ t_k\ -\ T_{\text{RL}}\ +\ \tau} \leq v_{\text{max}}.$$

Therefore, to ensure that there is always an optimal solution to $\mathcal{P}_1$:

$$\begin{cases}\tau \geq \dfrac{L_C\ -\ v_{\text{max}}(d_a T_{\text{RL}}\ +\ T_{\text{alert}})}{v_{\text{max}}}\\ 0 < \tau \leq T_{\text{RL}}\end{cases}. \tag{3.6}$$

Note that if $\frac{L_C}{v_{\max}} - (d_a T_{\text{RL}} + T_{\text{alert}}) < 0$, then $\mathcal{P}_1$ is feasible $\forall\ 0 < \tau \leq T_{\text{RL}}$. On the other hand, if $\frac{L_C}{v_{\max}} - (d_a T_{\text{RL}} + T_{\text{alert}}) \geq 0$, then $\mathcal{P}_1$ is feasible if and only if $0 < \frac{L_C}{v_{\max}} - (d_a T_{\text{RL}} + T_{\text{alert}}) \leq T_{\text{RL}}$.

The optimal solution of $\mathcal{P}_2$ always exists until $t_{\text{stop}} > t_l$, i.e., $t_{\text{stop}} - t_l > 0$. The $c(i,j)$ AV entering the CZ at t_l with $v_{i,j}(t_l) = v_{\max}$, in the worst case, has to decelerate with a constant $u_{i,j}(t) = -\frac{v_{\max}}{t_{\text{stop}} - t_l}$ in order to stop at $p_{i,j}(t_{\text{stop}}) = L_c - \delta$. Therefore, if $t_{\text{stop}} - t_l = \frac{2(L_C - \delta)}{v_{\max}} > 0$, then $\mathcal{P}_2$ is feasible $\forall\ t$.

The AV $c(i,j)$, which at time t_k is at a distance less than d_{follow} from the preceding $c(i-1,j)$ vehicle ($p_{i-1,j}(t_k) - p_{i,j}(t_k) < d_{\text{follow}}$), assumes the same dynamic behavior of the HDV (Section 3.1.4.1) from t_k onward throughout the journey.

3.1.4.3 Action-delayed RL

Henceforth, we assume that the dynamics of the vehicles is discretized with a sampling time T_S, while the intersection controller works at fixed TLBs, of $T_{\text{RL}} = nT_S$ duration, with $n \in \mathbb{N}$ fixed. Therefore, the amber traffic-light duration and the delay, already mentioned in Section 3.1.3, are respectively $T_{\text{alert}} = mT_S$, with $m < n \in \mathbb{N}$ fixed, and $T_{\text{delay}} = d_a T_{\text{RL}}$.

We characterize the decision process for urban intersection traffic-light control as a DDMDP [91] with constant action and cost delays denoted by the tuple $E_1 = (X, A, O, f, g, h, d_a, d_c)$ where d_a is the constant action delay, and $d_c = d_a$ the constant delay to observe the reward.

The controller at each k-th TLB observes the state of the system and has to select a control input that will be applied at the $k + d_a$-th TLB. This means that, at the k-th TLB the action selected at the $k - d_a$-th TLB is applied. The reward is measured at $k + d_a + 1$ TLB when the action is decided at the k-th TLB. As shown in [21], a DDMDP with action and cost delays is reducible to a classical MDP $E_1 = (I_X, A, O, f_I, g_I, h_a)$ without delay, where $I_X : X \times A^{d_a}$ is the "modified" state set, $f_I : I_X \times A \to I_X$ and $g_I : I_X \to O$ are respectively the "modified" transition and observation functions, while $h_a : X \times A^{d_a} \times A \to \mathbb{R}$ the "modified" reward function. Specifically, denoting by $i^{(k)} = (x^{(k)}, a^{(k-d_a)}, \dots, a^{(k-1)}) \in I_X$ the modified state at the k-th time instant, the modified reward function takes the form $r^{(k+1)} = h_a(i^{(k)}, a^{(k)}) = h(x^{(k)}, a^{(k-d_a)})$.

Note that the reward $r^{(k+1)}$ does not depend on the action at the k-th TLB but rather on the action decided at the $k - d_a$-th TLB $a^{(k-d_a)}$. Hereinafter we will refer to $\mathbb{N} \cup \{0\}$ as $\mathbb{N}_+$; and to $\mathbb{B} = \{0, 1\}$ as the Boolean domain.

State

The state $x^{(k)} \in X$ at the k-th time instant is equal to $q^{(k)}$, where $q^{(k)} \in {\mathbb{N}_+}^{n_l}$ is a vector in which the j-th element $q_j^{(k)}$ element represents the number of vehicles in the CZ of the j-th lane.

Action

The action $a^{(k)} \in A := \mathbb{B}^{n_l}$ at the k-th time instant is a vector having the number of elements equal to the number of lanes. At the k-th TLB, the traffic controller decides which lanes to be open, i.e., for which lanes the traffic-light will be green at the $k + d_a$-th TLB. If $a_j^{(k)} = 1$, the traffic-light will be green, if $a_j^{(k)} = 0$, the traffic-light will be red for lane j at the $k + d_a$-th TLB. Note that only those lanes which are non-conflicting can be open simultaneously. Therefore, we reduce the problem imposing that the actions of non-conflicting lanes are equal. Hence, $a_j^{(k)} = a_l^{(k)}$ if the pair (j, l) are non-conflicting. Thus, the action space can be reduced to only choosing elements for the set $\mathcal{L}$, i.e., the non-conflicting lanes.

Reward function

The traffic-intersection controller aims at minimizing the queue length at each lane while maximizing the outflow of vehicles at a given TLB. Hence, we consider the reward-function $h(x^{(k)}, a^{(k-d_a)}) = \| Wq^{(k)} \|_1 - \| Wq^{(k+1)} \|_1$, where $W \in \mathbb{R}^{n_l}$ is a weight vector. The weight vector W allows to assign to each lane a priority. If we want to impose a higher priority for the j-th lane, we will assign to the j-th element of W (w_j) a higher value than the others ($w_i < w_j,\ \forall i \neq j$). Imposing $W = [1, 1, \dots, 1]$ implies that there is the same priority for each lane in the optimization.

3.1.4.4 Optimal policy and Q-learning

The traffic-intersection controller has to learn an optimal π_1^*. We also assume that d_a decided actions are already awaiting execution at $k = 0$ which, along with $x^{(0)}$, constitute the initial modified state $i^{(0)}$. In the following, we rely on a L_1 tabular Q-Learning algorithm [180] in a non-episodic framework, in which the optimal policy now corresponds to:

$$\pi_1^*(i^{(k)}) = \arg\max_{a \in A} Q^* \left(i^{(k)}, a \right). \tag{3.7}$$

The reward inherently depends on the dynamics of the vehicles which have been described in Section 3.1.4.1 and Section 3.1.4.2, which in turn depend on the decision of the traffic intersection controller. The dynamics is non-linear and discontinuous as well. Being a model-free approach, the proposed method learns the optimal decision without using the model explicitly.

3.1.4.5 Implementation and results

We employ MATLAB for training and evaluating the proposed approach. We consider a MZ of size $L_M = 30\,\mathrm{m}$, the lengths of the CZ and the EZ are both equal to $400\,\mathrm{m}$. The maximum speed is set to $v_{\text{max}} = 13\,\mathrm{m\,s^{-1}}$.

The vehicles arrivals follow a Poisson process with an arrival rate of $1125\,\mathrm{veh\,h^{-1}}$ at each lane. Each vehicle's initial speed $v_{i,j}(t_0)$ is uniformly randomly sampled from $[9\,\mathrm{m\,s^{-1}}, 11\,\mathrm{m\,s^{-1}}]$, and an initial acceleration $u_{i,j}(t_0)$ is uniformly randomly sampled from $[0\,\mathrm{m\,s^{-2}}, 0.5\,\mathrm{m\,s^{-2}}]$. We set the capacity of the intersection equal to $N_{\text{max}} = 100$. We investigate three different scenarios where the fractions of the two kind of vehicles are respectively: 25% of AVs and 75% of HDVs; 50% of AVs and 50% of HDVs; 75% of AVs and 25% of HDVs. The jam-distance $s^0_{i,j}$ and the safety time-gap $T_{i,j}$ are set at $2\,\mathrm{m}$ and $5\,\mathrm{s}$ respectively for the IDM model. We take $\xi = 1.6\,\mathrm{m}$. Recall that when the traffic-intersection controller informs the AV that the next traffic-state is red, the AV stops at a δ-distance away from the intersection. We set $\delta = 20\,\mathrm{m}$. We set $d_{\text{follow}} = 50\,\mathrm{m}$.

The RL controller is trained according to a non-episodic Q-Learning problem and starts with an intersection having zero vehicles. Each simulation stops when the 2000-th vehicle enters the intersection. We set $W = [1, 1, \dots, 1]$, hence, the controller does not priorities among the lanes.

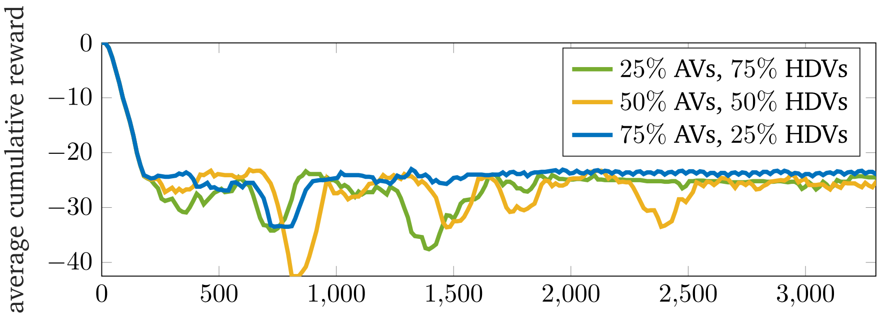


Figure 3.2: Average cumulative reward during simulations with a vehicles arrival rate of $1125\,\mathrm{veh\,h^{-1}}$ at each lane.

We performed all the simulations considering $T_{\text{RL}} = 15\,\mathrm{s}$, $T_{\text{alert}} = 3\,\mathrm{s}$, and $T_{\text{delay}} = 2T_{\text{RL}} = 30\,\mathrm{s}$, i.e., $d_a = 2$. We assume an ϵ-greedy policy with an exploration decay of $1/k$, with k number of performed decision steps. Moreover, following [22], we set $\gamma^k = c_1/(k + c_2)$, where c_1 and c_2 are both equal to 1.

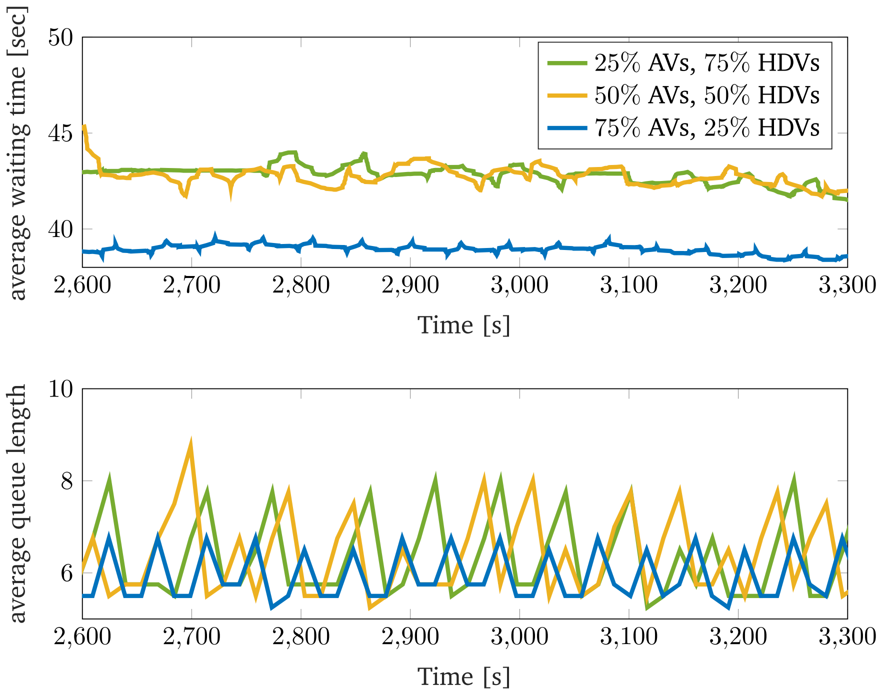


Figure 3.3: Average vehicles waiting time (top), and average queue length in the intersection lanes (bottom) starting at 2600 s, when the cumulative average reward settles down (see Figure 3.2), in all the simulations with a vehicles arrival rate of $1125\,\mathrm{veh\,h^{-1}}$ per lane.

In Figure 3.2, we compare the average cumulative reward in the performed experiments with a constant arrival rate of $1125\,\mathrm{veh\,h^{-1}}$ per lane. We can observe that a scenario with an higher value of AVs leads to a faster convergence and to an higher average cumulative reward.

In Figure 3.3 we can also observe that the average waiting time of vehicles, when the reward settles down, is lower in a scenario with 75% of AVs than in a scenario with 25% of AVs. Moreover, the average total queue length at the intersection assumes a pseudo-periodic behavior in the former scenario, while the queue still presents some undesirable steep hikes.

As expected, the clear advantage of introducing AVs can be observed also by comparing the energy consumption of the two different vehicles.

Table 3.1: Statistics of the $\int u^2(t)$ for coexisting AVs and HDVs

	mean	**median**	**mode**	**std dev**
25% AVs	4.541	1.947	0	9.379
75% HDVs	3.001e+03	83.511	1.317	8.597e+03
50% AVs	4.611	1.761	0	10.494
50% HDVs	6.118e+03	83.369	1.579	1.606e+04
75% AVs	6.745	1.532	0	15.709
25% HDVs	5.856e+03	83.215	0	1.322e+04

Indeed, as highlighted in Table 3.1, where for both vehicle types some statistical indices of the integral of the squared acceleration profiles over time are shown, AVs outperform HDVs.

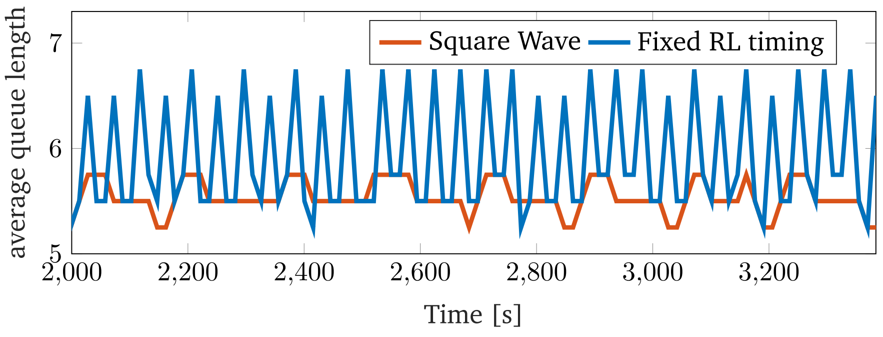


Figure 3.4: Comparison of the average queue length in the intersection lanes with 75% AVs performing the proposed approach (blue), and using a square wave traffic-light (orange) with a vehicles arrival rate of $1125\,\mathrm{veh\,h^{-1}}$ per lane. The time interval begins at $2000\,\mathrm{s}$, when the cumulative average reward settles down (see Figure 3.2).

Results obtained so far have highlighted how the AVs introduction in a urban intersection can lead to better controller performance and total energy consumption reduction. However, they do not point out what benefits are introduced by the proposed RL approach.

Clearly, the main advantage of RL in this context is its ability to learn a traffic management policy without an explicit mathematical formulation of the intersection dynamics which is not easy to model[1]. It consists of

[1]We remark that the model is used for simulation purpose only, while the learning algorithm has no knowledge of it.

inherent interaction between different dynamical units (i.e., vehicles), some of which have unpredictable behavior.

However, as shown in Figure 3.4 the proposed approach is failing to outperform a simple traffic light modeled by a delayed square wave with period equal to $2T_{RL}$ and the same d_a delay. In our belief, this unpleasant result is a consequence of the RL fixed timing operation setting. This choice, which we initially pursued for simplicity, does not allow the RL intersection controller to adopt a better performing behavior than a square wave traffic light with equal fixed timing. For this reason, in the following section, we present an event-driven RL approach, where RL timing is triggered by certain events occurring. The mentioned trigger events have been

3.1.5 The event-driven RL approach

In the following, we present the second approach proposed to address the traffic flow control problem of the urban intersection described in Section 3.1.3. Here, we assume that also in this case the traffic-intersection controller constantly observes and receives information about the current traffic condition in the intersection, but, differently from the former approach, triggers the decision-making process of a RL-based traffic-light controller when certain trigger conditions (TCs) are met[2]:

(TC1) at time t_k the CZ of j-th lane is empty ($C_j^{(t_k)} = \emptyset$) and the traffic-light status is green at the j-th lane;

(TC2) at time t_k a vehicle $c(i,j) \in C_A$ enters the intersection and the traffic-light status is green at lane j;

(TC3) a trigger did not occur for a T_{silence} time interval.

[2]The trigger conditions, along with the switching condition that will be introduced later, are design choices which have the twofold purpose of (1) allowing the RL to avoid trivial situations (e.g., empty lanes with corresponding green traffic lights), and (2) steering the RL to create queues led by autonomous vehicles.

Roughly speaking, in this case the duration of each TLB is variable and depends on the inter-time between two successive triggering events. In details, with condition (TC1) we are enabling the traffic-light controller to possibly close empty lanes. Condition (TC3), instead, is particularly useful in traffic intersections with a low percentage of AVs, and highly congested traffic. Specifically, if the traffic-light status has not been changed for a long-time, the condition (TC3) will ensure that the controller would try to see whether it is required to change the status. The motivation behind condition (TC2) is less intuitive and needs the introduction of other notations and assumptions, provided in the following.

Definition 3. A vehicle $c(i,j) \in C_A$ in the CZ of the j-th lane is the *leader* vehicle of the j-th lane if $\nexists$ $c(k,j) \in C$ front vehicle in the CZ of $c(i,j)$ (cf. Definition 1).

Definition 4. A vehicle $c(i,j) \in C_A$ in the CZ of the j-th lane is a *follower* vehicle if there exists $c(k,j) \in C$ front vehicle in the CZ of $c(i,j)$ (cf. Definition 1).

We denote by $\phi^j(t)$ the traffic-light status at time t for the j-th lane imposed by a traffic-light controller. $\phi^j(t) = 1$ indicates that the traffic-light is green, while $\phi^j(t) = 0$ means that the traffic-light is red. We impose a $\phi^j(t) = -1$ condition, corresponding to a yellow traffic-light, of fixed duration T_{alert} each time that a traffic-light switch occurs. When the traffic-light controller is triggered, it selects the future traffic-light status ϕ^j_{new} for each lane j, and enables a *sleep mode* of T_{sleep} during which no triggers can occur. T_{silence} is counted after a trigger occurrence.

We assume that ϕ^j_{new} will be applied only d_a seconds after the t_k at which it is chosen, i.e., $\phi^j(t_k + d_a) = \phi^j_{new}$. In general, we impose $d_a = T_{\text{sleep}}$. A different d_a is considered if the following switch condition (SC) occurs:

(SC1) If at time t_k a vehicle $c(i,j) \in C_A$ enters the intersection and the traffic-light status is green at lane j (condition (TC2)), and if the future traffic-light status is $\phi^j_{new} = 0$, then the traffic-light becomes red only when the vehicle $c(i-1,j)$ (front vehicle of $c(i,j)$) enters the MZ.

This last condition ensures that the at $t^m_{i-1,j}$, time instant at which the vehicle $c(i-1,j)$ crosses the MZ, ϕ^j_{new} is applied[3], i.e., $d_a = t^m_{i-1,j} - t_k$. Moreover, starting from $t^m_{i-1,j}$, the $c(i,j)$ vehicle is promoted leader vehicle of the j-th lane. When the traffic-controller again informs the AV $c(i,j)$ of a future green traffic-light, the vehicle can schedule an optimal acceleration profile, approaching the MZ as soon as the traffic-light turns green, thus increasing the throughput.

As in Section 3.1.4, the ultimate goal of the RL-based traffic-light controller is to minimize the total number of vehicles queuing at the intersection, thus maximizing the rate of vehicles outflow. We assume to model the HDV as in Section 3.1.4.1, while the AVs dynamics is defined in the following section.

3.1.5.1 Autonomous vehicle

When $c(i,j) \in C_A$ enters the intersection it can assume the role of leader or follower according to Definition 3 and 4.

If at time $t = t_k$ the traffic-intersection controller selects $\phi^j_{\text{new}} = 0$, the leader AV schedules a uniform deceleration profile $u_{i,j}(t)$ leading to stop its cruise δ distance away from L_C.

Conversely, if at time $t = t_k$ the traffic-intersection controller selects $\phi^j_{\text{new}} = 1$, the leader AV in $p_{i,j}(t_k) = p_k$ with $v_{i,j}(t_k) = v_k$ solves the following optimization problem:

[3]Note that no triggers can occur before that ϕ^j_{new} is applied

$$\begin{aligned}
&\mathcal{P}_3 : \min_{u_{i,j}(\cdot)} \frac{1}{2}\int_{t_k}^{t_{i,j}^m} u_{i,j}^2(t)dt,\\
\text{subject to: } &\dot{v}_{i,j}(t) = u_{i,j}(t)\ \dot{p}_{i,j}(t) = v_{i,j}(t)\\
&p_{i,j}(t_k) = p_k,\ v_{i,j}(t_k) = v_k,\\
&v_{i,j}(t) \le v_{\text{max}},\ v_{i,j}(t) \ge 0\ \forall\ t\ \in \left[t_{i,j}^0, t_{i,j}^m\right],\\
&t_{i,j}^m \ge t_k + d_a + T_{\text{alert}},\\
&t_{i,j}^m \le t_k + 2d_a + T_{\text{alert}}\\
&p_{i,j}(t_{i,j}^m) = L_C.
\end{aligned} \tag{3.8}$$

The optimization problem is similar to the one in (3.4), the only difference is related to the $t_{i,j}^m$ constraints. Indeed, while in the previous problem we assumed a fixed delay $T_{\text{delay}} + T_{\text{alert}}$, in this case our delay strictly depends on d_a that, as described in Section 3.1.5, has a variable value. Also in this case we relax the optimization problem by relying on a penalty function approach:

$$\begin{aligned}
&K_{v_{\text{max}}}\{\max(0,\ (v_{i,j}(t)\ -\ v_{\text{max}}))\},\\
&K_{v_{\text{min}}}\{\max(0,\ -v_{i,j}(t))\},\\
&K_{t_{i,j}^m}^1\{\max(0, (t_k\ +\ d_a\ +\ T_{\text{alert}})\ -\ t_{i,j}^m)\},\\
&K_{t_{i,j}^m}^2\{\max(0, t_{i,j}^m\ -\ (t_k\ +\ 2d_a\ +\ T_{\text{alert}})\}.
\end{aligned}$$

For $t > t_{i,j}^m$ the vehicles are assumed to travel with the maximum speed $v_{i,j}(t) = v_{\text{max}}$.

In order to ensure the feasibility of $\mathcal{P}_3$, d_a must be properly chosen. All the above definitions hold either (1) when an AV enters the intersection at $t = t_k$, initial time of a the k-th TLB, or (2) during the AVs cruise. If the AV $c(i,j)$ enters the intersection at $t = t_l$ with $t_k \le t_l < t_k + d_a$, the $\mathcal{P}_3$ feasibility depends on $\tau = t_k + d_a - t_l$. The worst condition occurs when at $t \gtrapprox t_k + d_a$ a new trigger selects a new traffic-light[4] that will be applied at $t \gtrapprox t_k + 2d_a + T_{\text{alert}}$. In this case the AV $c(i,j)$ should be able to cover a distance L_C at most at v_{max} within $[t_l,\ t_k + 2d_a + T_{\text{alert}}]$:

[4]We highlight that due to the T_{sleep}, t cannot be equal to $t_k + d_a$.

$$\frac{L_C}{t_k + 2d_a + T_{\text{alert}} - (t_k + d_a - \tau)} \leq v_{\max}.$$

As τ tends to 0 the above expression leads to the feasibility constraint $d_a + T_{\text{alert}} \geq \frac{L_C}{v_{\max}}$.

For those cases in which at the h-th lane, conflicting with the j-th lane, the condition (TC2) is met for the vehicle $c(q, h)$ and $\phi^h_{new} = 0$ (condition (SC1)), the crossing time $t^m_{i,j}$ of the vehicle $c(i, j)$, entering the j-th lane after the (SC1) occurrence, depends on $t^m_{q-1,h}$, i.e., $t^m_{i,j} \geq t^m_{q-1,h} + T_{\text{alert}}$. $t^m_{q-1,h}$ is not known a priori. Thus, no guarantees can be given about the feasibility of $\mathcal{P}_3$ in this scenario. However, without loss of generality, we can assume that a lane with a traffic light red has a null probability to have an empty CZ, thus the AV $c(i, j)$ has a null probability to be the leader of the j-th lane.

Note that an already scheduled AV will not update its profile if it receives a new traffic light controller action equal to the one of the previous scheduling.

A vehicle $c(i, j) \in C_A$ triggering condition (TC2) and informed of a $\phi^j_{\text{new}} = 0$ ((SC1) occurrence) is treated as a leader and schedules a decelerating profile that imposes its stop δ distance away from L_C. If at t_k it is informed of a future green light status $\phi^j(t_k + d_a) = 1$, it will solve the optimization problem $\mathcal{P}_3$.

In all the remaining scenarios, the dynamics of AVs follow the IDM model behavior presented in Section 3.1.4.1. This assumption takes place also when the sleep mode of T_{sleep} duration is active due to a previous trigger occurrence.

3.1.5.2 Action-delayed RL

Again we model the decision process for the urban intersection traffic-light controller as a discrete-time DDMDP [91] with action delays. Henceforth, we assume that the dynamics of the vehicles are discretized with a sampling time T_S, while the traffic-light controller is triggered by the events described in (TC1), (TC2), and (TC3). Hence the RL agent results in an event-driven discrete time controller.

We denote by t_{k-1} and t_k two successive time at which two different events occur. The DDMDP is also in this case a tuple $E_1 = (X, A, O, f, g, h, d_a, d_c)$, however in this case the action delay d_a follows the behavior described in Section 3.1.5, while the reward of an action chosen at $t = t_{k-1}$ is observed with a delay $d_c = 0$, due to the fact that no triggers are allowed within T_{sleep}. Here we consider a constant d_a except in the case in which Assumption (SC1) occurs. In this later case, d_a depends on the time at which the front vehicle of the AV meeting condition (TC2) enters the MZ.

We use the following notations:

- $x^{(t_k)} \in X$ the state of the RL system at t_k;
- $a^{(t_k)} \in A$ the RL control input (*action*) at t_k;
- $i^{(t_k)} = \left(x^{(t_k)}, a^{(t_{k-1}+d_a)}\right) \in I$ the information needed for optimal action selection at t_k;
- $g\left(x^{(t_k)}, a^{(t_{k-1}+d_a)}\right)$ the reward function.

Note that the reward function at t_k does not depend on the action at the same time instant, but rather on the action decided at the preceding trigger instant t_{k-1} and applied at $t_{k-1} + d_a$.

We perform also in this case a tabular Q-Learning algorithm with the same definition of state, action and reward provided in Section 3.1.4.3.

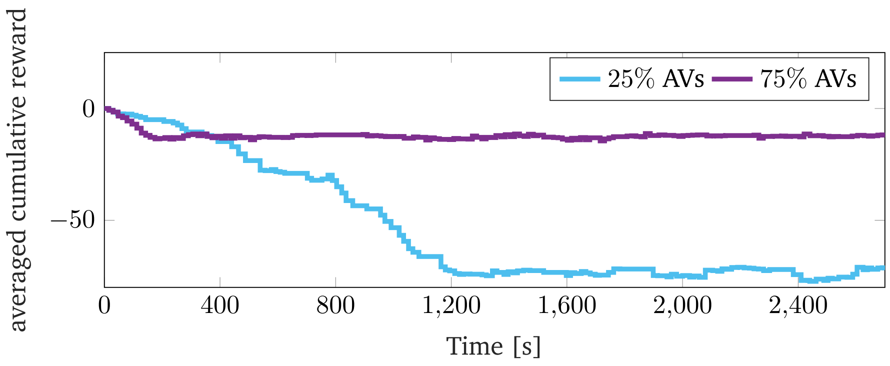
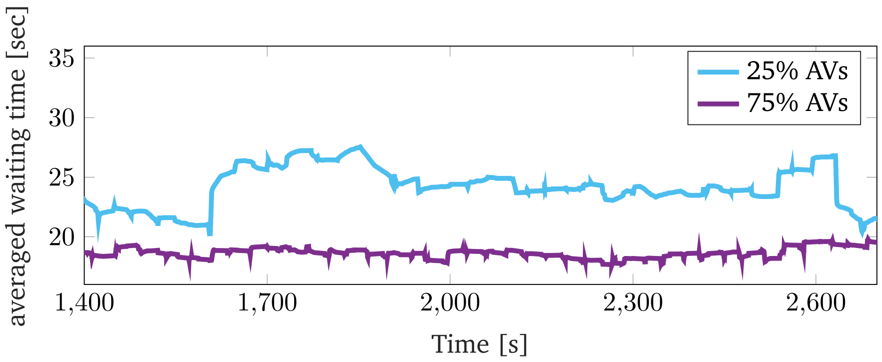


Figure 3.5: Averaged cumulative reward during simulations with a vehicles arrival rate of $1125\,\mathrm{veh\,h^{-1}}$ per lane.

3.1.5.3 Implementation and results

To evaluate the proposed approach we use the same MATLAB framework of Section 3.1.4.5 with $L_C = 200\,\mathrm{m}$. We investigate two different scenarios where the fractions of the two kind of vehicles are: (1) 25% of AVs and 75% of HDVs; (2) 75% of AVs and 25% of HDVs. We perform the experiments with an arrival rate of $1125\,\mathrm{veh\,h^{-1}}$ at each lane. We set $T_{\text{silence}} = 30\,\mathrm{s}$, and $T_{\text{sleep}} = 15\,\mathrm{s}$. Also in this case the RL controller is trained according to an infinite horizon Q-Learning problem. The simulation starts with an intersection having zero vehicles, and simulation stops when the 2000-th vehicle enters the intersection. We consider $T_{\text{alert}} = 3\,\mathrm{s}$. We assume the same RL agent training settings of the Section 3.1.4.5.

In Figure 3.5 we compare the averaged cumulative reward in the performed experiments with a constant arrival rate of $1125\,\mathrm{veh\,h^{-1}}$ per lane. We can observe that a scenario with an higher value of AVs leads to a faster convergence and to a higher averaged cumulative reward. In Figure 3.6 we can also observe that the average waiting time of vehicles, when the reward settles down, is lower in a scenario with 75% of AVs than in a scenario with 25% of AVs. Moreover, the averaged total queue length at the intersection assumes a pseudo-periodic behavior in the former scenario, while the queue still presents some undesirable steep hikes when the AV penetration rate is lower.



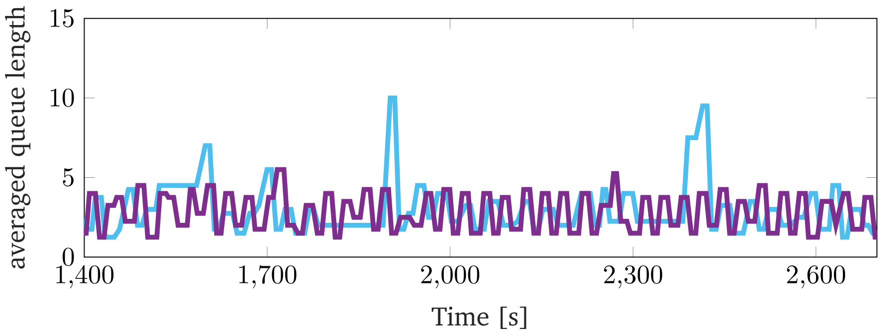


Figure 3.6: Averaged vehicles waiting time (top), and averaged queue length in the intersection lanes (bottom) during simulations with a vehicles arrival rate of $1125\,\mathrm{veh\,h^{-1}}$ per lane, starting at $1400\,\mathrm{s}$, when the cumulative average reward settles down (see Figure 3.5).

In Figure 3.7, we compare the results obtained applying the proposed approach with those obtained performing the approach of Section 3.1.4.5. The proposed approach appears to be more effective in the management of the queue length. The reason is that now, the traffic-light cycle duration can be adapted based on the nature of the vehicle. For example, when the event is triggered because of the condition (TC2) and the traffic-controller decides to select the red-light, the red-light will be switched only after all the vehicles preceding to the newly entered AV enter the intersection. Thus, the traffic-light cycle duration is adapted based on the nature of the vehicle and can be of variable length unlike in Section 3.1.4. Further, in this new scenario, the traffic-intersection controller coordinates with that AV as it would become the leader, whereas such a provision was not there in Section 3.1.4. Moreover, as shown in Figure 3.8 the proposed approach

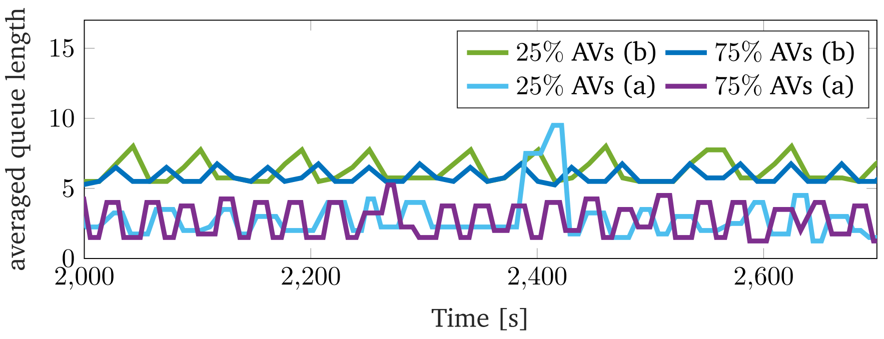


Figure 3.7: Comparison of the averaged queue length in the intersection lanes using (a) the proposed approach, and (b) the approach described in Section 3.1.4, with a vehicles arrival rate of $1125\,\mathrm{veh\,h^{-1}}$ per lane.

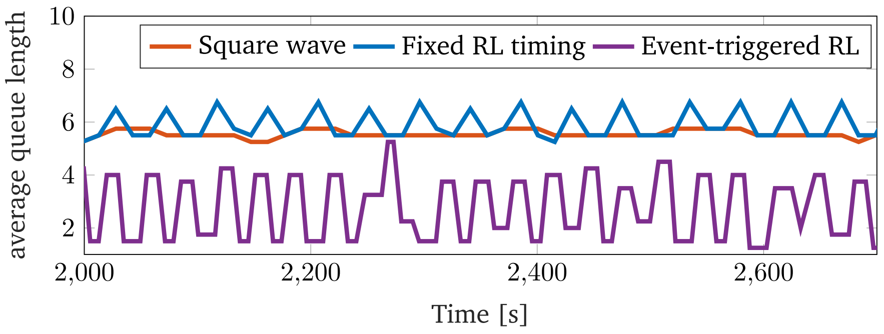


Figure 3.8: Comparison of the average queue length in the intersection lanes with 75% AVs performing the approach of Section 3.1.4 (blue), a square wave traffic-light (orange), and the proposed approach (purple) with a vehicles arrival rates of $1125\,\mathrm{veh\,h^{-1}}$ per lane. The time interval begins at $2000\,\mathrm{s}$, when the cumulative average reward settles down(see Figure 3.5).

outperforms the same simple traffic light of Figure 3.4, thus highlighting the potential of this latter proposed approach.

3.2 Intensity control of the FERMI seeded free-electron laser

We now move on to the second case study of the chapter. We consider the FERMI facility at Elettra Sincrotrone Trieste. We are dealing with a seeded Free-Electron Laser (FEL) [187, 9, 8, 10], in which the temporal and transverse overlap of the electron and laser beams are the most critical conditions to be achieved. In particular, while the temporal alignment between the two beams is controlled by a single actuator (a mechanical delay line), the transverse alignment depends on the transverse position of each beam at the input and output of the modulator undulator, i.e., the first magnetic structure used for electron beam energy modulation. Therefore, we first focus on designing a control strategy π_2^{1*} for *performance optimization*, thus bringing the FERMI facility to an optimal operating point, i.e., the one that guarantees the superposition of electrons and laser beams, starting from random initial conditions. Next, we consider a scenario in which an already optimized facility is subjected to changes in operating conditions, or machine drifts, such that it loses its optimality. In this case, we design a *performance recovery* control procedure π_2^{2*} that restores the machine operating condition to its optimum in the shortest possible time.

We provide a model of the FERMI environment E_2. However, we assume that f and g (Equations (2.1) and (2.2)) are unknown to the controllers. Therefore, both $\pi_2^{1^*}$ and $\pi_2^{2^*}$ must be learned in a model-free fashion.

We apply two different techniques directly on the FERMI: (1) the episodic Q-learning with linear function approximation L_2^1, for performance optimization, and (2) the continuous Natural Policy Gradient REINFORCE algorithm L_2^2, for performance recovery. Despite the simplicity of these approaches, we report satisfactory results.

3.2.1 Motivations

Optimal tuning of particle accelerators is a challenging task. These are extremely complex facilities, characterized by the absence of a model that accurately describes their dynamics, and by an often persistent noise which, along with machine drifts, affects their behavior in unpredictable ways. During standard operations, the optimal position recovery of the seed laser is continuously carried out by an automatic process [63] able to properly change the seed laser pointing on the basis of the output FEL intensity. This approach assumes that there is a correlation between the laser position and the FEL output. The correlation is low in correspondence of the optimal device performance. Thus, minimizing the correlation leads to FEL performance maximization. In case the natural jitter is not sufficient to determine the right direction to move the pointing of the seed laser, the system mimics the Extremum Seeking algorithm [14] by introducing an artificial noise which speeds up the convergence but, at the same time, affects the quality of the FEL radiation. This kind of model-free optimization techniques (e.g., Gradient Ascent and Extremum Seeking [14, 29]) are widely used in FEL facilities, but have some intrinsic disadvantages: (1) the need to evaluate the gradient of the objective function, which can be difficult to estimate when the starting point is far from the optimum, (2) the difficulty to determine the hyper-parameters, whose appropriate values depend on the environment and the noise of the system, and (3) the lack of "memory" to exploit the past experience.

3.2.2 Related literature

Several approaches have been proposed for FEL performance optimization, some of them being just as a proof of principle, others being actually employed [166]. A model-free approach using Gradient Ascent and Extremum Seeking algorithms has been investigated on the FERMI FEL at Elettra Sincrotrone Trieste [29]. Furthermore, a multi-physics simulation tool kit called OCELOT [6] has been designed at the European XFEL in Hamburg for the study of FELs and synchrotron light sources. Some generic optimization algorithms such as Extremum Seeking, Nelder Mead, and Bayesian optimization based on Gaussian processes are already implemented in the framework. Gaussian process and Bayesian optimization have been also employed to tune the quadrupole currents at the Stanford Linear Accelerator Center (SLAC) [110, 109] and to optimize the self-amplification power of the spontaneous emission of the Free electron LASer in Hamburg (FLASH) at the Deutsches Elektronen-SYnchrotron (DESY) [7]. In recent years, Machine Learning (ML) techniques have led to many improvements and successful implementations in the field of particle accelerators, from automated alignment of various devices with beam, to optimizing different parameters, see for example [140, 57, 54, 62, 17, 118]. An overview of the opportunities provided by the application of ML for particle physics is given in [53]. In particular, the authors of [86, 141] consider Reinforcement Learning (RL) for control and performance improvement. In [52], the authors advocate the use of artificial neural networks to model and control particle accelerators in combination with RL. Moreover, recent works [55, 56, 75, 28, 27, 126] have presented RL methods used in the context of FELs.

3.2.3 FEL alignment system

Despite the complex structure of the FERMI, the alignment process can be described by the simple setup shown in Figure 3.9. Assuming a stable electron beam trajectory, the superimposition of the two beams is achieved by imposing a specific laser trajectory. In particular, by properly tuning the angle of two tip-tilt mirrors upstream of the facility (TT1 and TT2), a coherent optical radiation downstream of the chain is detectable by the intensity sensor (I_0 monitor). For a detailed description of the alignment process, and of each of the devices shown in Figure 3.9, we refer readers to [27, 28]. Here, we only point-out that the intensity detected by the I_0 monitor can be adjusted by properly controlling the pitch and yaw movements of the tip-tilts, which depend on the voltage regulation of some piezomotors (two for each tip-tilt mirror).

We denote by $v^{(k)}_{\text{pitch},i}$ and $v^{(k)}_{\text{yaw},i}$ the servo motors voltages at the k-th time instant, governing respectively the pitch and yaw angles of the i-th tip-tilt mirror. The angles are controlled incrementally, i.e., at each k-th time instant, the i-th tip-tilt mirror receives the pitch and yaw

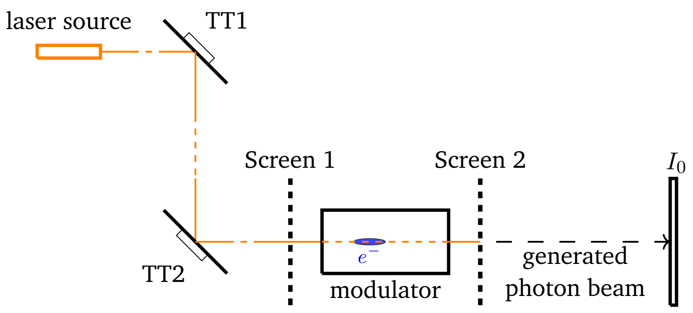


Figure 3.9: Simple scheme of the FERMI FEL seed laser alignment set up. TT1 and TT2 are the tip-tilt mirrors, Screen 1 and Screen 2 are two removable Yttrium Aluminum Garnet (YAG) screens with Charge Coupled Devices (CCDs), and I_0 is the employed intensity sensor monitor.

displacements as input (respectively denoted by $\delta v^{(k)}_{\text{pitch},i}$ and $\delta v^{(k)}_{\text{yaw},i}$), and reaches the new $v^{(k+1)}_{\text{pitch},i}$ and $v^{(k+1)}_{\text{yaw},i}$ according to:

$$\begin{aligned} v^{(k+1)}_{\text{pitch},i} &= v^{(k)}_{\text{pitch},i} + \delta v^{(k)}_{\text{pitch},i} \\ v^{(k+1)}_{\text{yaw},i} &= v^{(k)}_{\text{yaw},i} + \delta v^{(k)}_{\text{yaw},i} \end{aligned}. \tag{3.9}$$

Let Z, U and I be respectively the state set, the control set and the output set of the system represented in Figure 3.9. We model it as a discrete-time dynamical system $\mathcal{S}$ whose dynamics follows:

$$\begin{cases} z^{(k+1)} &= Az^{(k)} + Bu^{(k)} \\ I^{(k)} &= f\left(z^{(k)}\right) \end{cases}, \tag{3.10}$$

where $z^{(k)} := \left[v^{(k)}_{\text{pitch},1}\ v^{(k)}_{\text{yaw},1}\ v^{(k)}_{\text{pitch2}}\ v^{(k)}_{\text{yaw},2}\right]^\top \in Z \subset \mathbb{R}^4$ is the state at the k-th time instant, $u^{(k)} := \left[\delta v^{(k)}_{\text{pitch},1},\ \delta v^{(k)}_{\text{yaw},1},\ \delta v^{(k)}_{\text{pitch},2},\ \delta v^{(k)}_{\text{yaw},2}\right]^\top \in U \subset \mathbb{R}^4$ is the control input at the k-th time instant, $I^{(k)} \in I \subset \mathbb{R}$ is the detected intensity at the k-th time instant, $A, B \in \mathbb{R}^4 \times \mathbb{R}^4$ are two identity matrices, and $f : Z \to I$ is a non linear function. In other words, it is a Wiener system [121], i.e., a model consisting of a nonlinear static element preceded by a linear dynamical system. Clearly, the dynamical system $\mathcal{S}$ (Equation (3.10)) is only a simplified approximation of the FERMI dynamics, and is provided here to capture the relevant behavior of the dynamics to be controlled. Given the model-free strategies of the employed RL algorithms, directly applied to the facility, however, $\mathcal{S}$ will not be used.

3.2.4 Decision making strategy

We characterize with $E_2 = (X, A, O, f, g, h)$ the alignment process of the FERMI, where we set $f = g$. At each k-th time instant the controller observes the current voltage applied to each piezomotor, and selects the respective displacement of each piezomotor leading to a detected intensity by the I_0 monitor as close as possible to a target one I_T.

State

The state $x^{(k)} \in X$ at the k-th time instant is equal to $z^{(k)}$ in Equation (3.10), and is directly provided by the two tip-tilts mirrors of Figure 3.9. It can assume values that satisfy the physical constraints of the piezo-motors [39]:

$$x_{\text{MIN}} \leq x^{(k)} \leq x_{\text{MAX}}, \ \forall k \tag{3.11}$$

Action

The action $a^{(k)} \in A$ at the k-th time instant is equal to $u^{(k)}$ in Equation (3.10). We allow inputs for which the component-wise inequality (Equation (3.11)) is not violated.

Reward function

Two different reward functions are formulated to address, respectively, the optimal working point control problem, and the recovery of the optimal tuning of the facility. In the former, the reward is shaped according to [123]:

$$h(x^{(k)}, a^{(k)}) = \bar{r} + \zeta \cdot \frac{\xi\, I^{(k+1)} - I^{(k)}}{I_T}, \tag{3.12}$$

where $\bar{r}$ is taken equal to 1 if the target is reached (0 otherwise), while the values of $\xi > 0$ and $\zeta > 0$ are set empirically. The specific design of Equation (3.12) allows to reward the agent in correspondence of state-action pairs that lead to a sufficiently increased detected intensity $\xi I^{(k+1)} > I^{(k)}$, and to penalize it otherwise.

In the recovery of the optimal working point control problem, the reward is instead computed according to:

$$h(x^{(k)}, a^{(k)}) = \frac{I^{(k+1)}}{I_T} - 1. \tag{3.13}$$

When $I^{(k+1)}$ vanishes, $\frac{I^{(k+1)}}{I_T}$ vanishes as well, and the agent receives a penalty of -1. Conversely, when $I^{(k+1)}$ approaches I_T, $\frac{I^{(k+1)}}{I_T}$ approaches 1, and the agent receives a reward of 0. In other words, we want to reward the agent in correspondence of state-action pairs leading to a detected intensity downstream the facility as close as possible to I_T.

3.2.4.1 Optimal policies

In the present application, we have to learn two different optimal policies π_2^{1*} and π_2^{2*}.

For the attainment of an optimal working point (π_2^{1*}) we rely on a L_2^1 Q-learning algorithm in an episodic framework resulting in an optimal policy:

$$\pi_2^{1*}(x^{(k)}) = \arg\max_{a \in A} Q^*(x^{(k)}, a). \tag{3.14}$$

The choice of the Q-learning among other RL approaches is due to its simplicity and the fact that the problem admits a non-sparse reward which is beneficial for speeding up the learning [123]. In order to work with a continuous state space, we rely on a function approximation version of the Q-Learning. We assume that the actions are finite $a \in A = a_1, ..., a_N$, thus resulting in an action-value function that can be represented as a collection of maps $Q(x, a_1), ..., Q(x, a_N)$. We parametrize each $Q(x, a_j)$ as $Q(x, a_j) = \theta_j^T \varphi(x)$, where $\varphi(x)$ is a vector of features, and θ_j a weight vector associated to the j-th input a_j. Thus, the whole $Q(x, a)$ is specified by the vector of parameters $\theta = [\theta_1^\top, \dots, \theta_N^\top]^\top$. In particular, we employ Gaussian Radial Basis Functions (RBFs) as features; i.e., given a set of centers $\{c_i \in X, \quad i = 1 \dots, d\}$, we set $\varphi(x) = [\varphi_1(x), \dots, \varphi_d(x)]^\top$, in which $\varphi_i(x) : \mathbb{R}^n \to \mathbb{R}$ is:

$$\varphi_i(x) = \exp\left(-\frac{\|x - c_i\|^2}{2\sigma_i^2}\right), \tag{3.15}$$

where σ_i determines the decay rate of RBF and n is the dimension of the state vector.

For the performance recovery (π_2^{2*}), we apply a policy gradient (PG) strategy, i.e., the L_2^2 REINFORCE algorithm [183], which aims at finding the optimal $\theta^* \in \Theta$, solution of the optimization problem

$$\max_{\pi} \mathbb{E}\left[J_\pi(\xi)\right], \tag{3.16}$$

where $\xi = (x^{(0)}, a^{(0)}, x^{(1)}, a^{(1)}, \dots, a^{(T-1)}, x^{(T)})$ is a state-input trajectory obtained by following a particular π, and $J_\pi(\xi) = \sum_{k=0}^{T-1} h(x^{(k)}, a^{(k)})$; $\forall (x^{(k)}, a^{(k)}) \in \xi$ is the corresponding cumulative reward. The trajectory ξ can be thought of as a random variable that has a probability distribution $P(\xi|\theta)$. This result is achieved by updating θ along the gradient of the objective function. The resulting optimal policy is:

$$\pi_2^{2*} = \arg\max_{\theta \in \Theta} \mathbb{E}\left[J_{\pi_{2_\theta}}(\xi)\right]. \tag{3.17}$$

Here we consider an agent consisting of four independent parametrized policies, one for each element of the action vector ($a_i^{(k)}$, $i \in \{1,2,3,4\}$), which are shaped according to the Von Mises distribution[5]:

$$\pi_i(a_i^{(k)}|x^{(k)};\theta_i) = \frac{e^{\psi_i cos(a_i^{(k)}-\mu_i)}}{2\pi\mathcal{I}_0(\psi_i)} \quad \text{s.t., } i \in \{1,2,3,4\},$$

where $\psi_i = e^{\phi_i}$ is a concentration measure, μ_i is the mean, $\mathcal{I}_0(\psi_i)$ is the modified Bessel function of the first kind [3] and $\theta_i = [\mu_i, \phi_i]$ is the i-th policy parameter vector, updated at each step of the procedure.

In order to reduce the variance of the gradient estimates, typical of the PG approaches [190], we employ a natural PG (NPG) version [87] of the REINFORCE algorithm, in which a linear transformation of the gradient is adopted by using the inverse Fisher information matrix $F^{-1}(\theta)$.

For more details about the algorithms, and for the pseudo-codes, see [27].

3.2.4.2 Implementation and results

In the following, we describe the experimental protocols, and we report the results obtained for both the two considered tasks.

Optimal working point attainment problem

The problem of defining a policy, able to lead the plant to an optimal working point starting from random initial conditions, requires to split the experiments in two phases: (i) a training, which allows the controller to learn a proper policy, and (ii) a test, to validate the ability of the learned policy to properly behave, possibly in conditions not experienced during training.

[5]such a distribution is a convenient choice when the state and action spaces are bounded, since it is null outside a bounded region

The training consists of 300 episodes. The number of episodes has been chosen after preliminary experiments on a device simulator. At the beginning of the training the target value I_T is selected. It remains the same for all the training episodes. At each time step k, the action provided by the agent is applied, and the new intensity $I^{(k+1)}$ is compared with the I_T. The episode ends in two cases: (1) when the detected intensity in the new state $I^{(k+1)}$ is greater than or equal to a certain fraction of p_T of the target ($p_T I_T$); (2) when the maximum number of allowed time steps is reached. When the first condition occurs, the goal is achieved.

We performed a RBF approximation version of Q-learning with an ϵ-greedy policy. During the training procedure the exploration ϵ, and the learning rate α, decay according to the following rules [176, 64]:

$$\alpha \leftarrow \alpha \cdot \frac{N_0 + 1}{N_0 + \#\text{episode}}, \quad \epsilon \leftarrow \frac{1}{\#\text{episode}}; \tag{3.18}$$

where the $N_0 = 20$, while $\alpha = 0.1$ is the initial value of the learning rate. We set $\gamma = 0.99$, $\xi = 0.99$, $p_T = 90\%$, and $\sigma^2 = 0.0075$ for each RBF. We consider that each episode can consist at most of 10000 steps.

At the end of each episode a new one begins from a new initial state, randomly selected, until the maximum number of episodes is reached. Then, a test (with random initial states) of 50 episodes is carried out for the same target condition of the training, but with a fixed $\epsilon = 0.05$ as in [113].

The number of time-steps per episode for the whole training phase is reported in Figure 3.10. The steep decrease of the number of time steps shows that a few episodes are sufficient to get a performance close to the one obtained after a whole training phase. In other words, the exploration carried out during the first episodes provides a valuable information for the estimation of the Q-function and, as a consequence, of an appropriate policy. We believe that the main reason is the effectiveness of the reward shaping (Equation (3.12)), that allows to reward the agent at each time step, as opposite of a sparse reward occurring only at the end of the episodes. Such a shaping seems reasonable for the problem at hand, and is based on the assumption that the observed intensity change of two subsequent steps is significant for guiding the learning.

The number of time-steps per episode during the test phase is visible in Figure 3.11 and is consistent with the training results.

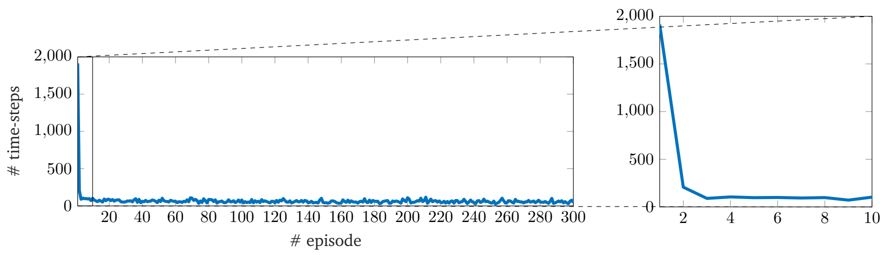


Figure 3.10: Number of time-steps for each episode during a single run of training performed on the FERMI FEL system. The number of time-steps required in the first 10 episodes is highlighted in the enlarged portion.

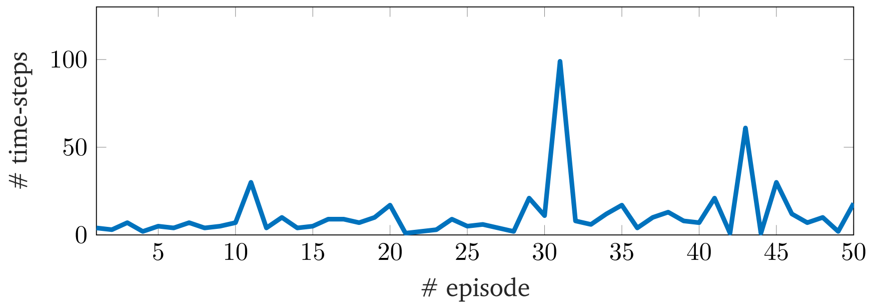


Figure 3.11: Number of time-steps for each episode during a single run of test performed on the FERMI FEL system.

However, during the test phase, we observe some unsuccessful trials. Although some further investigation is needed, it might be due to either (i) the occurrence of unexpected drifts of the target during the test, or (ii) the discrete set of actions employed, consisting of fixed steps that can prevent reaching the goal, starting from random initial conditions.

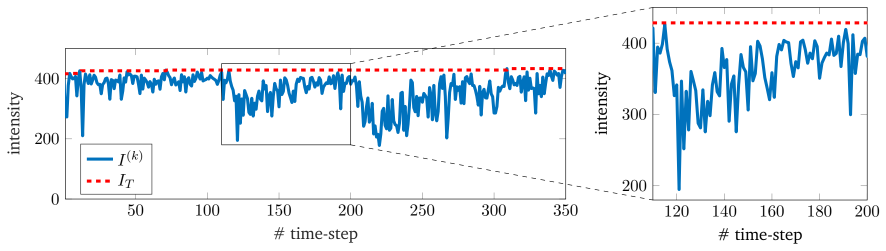


Figure 3.12: Intensity during a single run of NPG REINFORCE on FEL. The blue line represents the detected intensity. The dashed red line represents the target intensity. The target intensity is almost constant during the whole run. Two perturbations have been manually introduced by moving the coarse motors. The first perturbation and subsequent recovery are highlighted in the enlarged portion.

Recovery of optimal working point

In the recovery of optimal working point, we look for a controller able to quickly and properly adapt its policy to thermal drifts or wavelength variations, which result in a displacement of the optimal working point. Here, we want to employ the learning as an adaptive mechanism, to face the machine drifts. Thus, in this case, a test phase would be meaningless, since adaptation occurs during learning only. We perform the NPG REINFORCE algorithm, which is able to work with a continuous action space and, thus, to allow for precise fine tuning of the facility.

The experiment consists of a single training phase, at the beginning of which, the system is set on an optimal working point (including both the state and the I_T), manually found by experts. During the experiment, some misalignment are forced by manually changing the coarse motors position.

Each time that $I^{(k+1)}$ results greater than I_T, the target intensity is updated according to:

$$I_T \leftarrow I_T + 0.1(I^{(k+1)} - I_T). \tag{3.19}$$

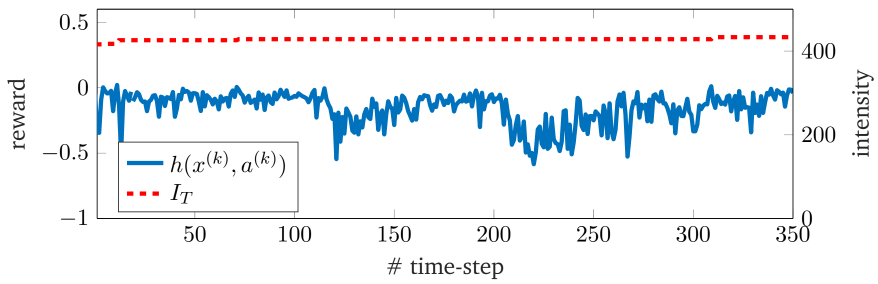


Figure 3.13: Reward (blue) and target intensity (red, dashed) during a single run of NPG REINFORCE on FEL. The slight increases of target intensity correspond to positive rewards.

Figure 3.12 and Figure 3.13 report the detected intensity and the reward, together with the target, during the experiment. The target intensity is not significantly updated since the system is initialized on an optimal working point. Two drift events took place, the first around time-step 120, and the second around time-step 210.

Both plots, (detected intensity and reward), clearly show the capability to recover the optimal pointing of the laser. Moreover, it is possible to observe how the algorithm quickly replies to disturbances of the environment settings (marked by negative reward spikes).

3.3 Discussion

In this chapter, we dealt with the application of RL to two real-world dynamical systems, namely a mixed autonomy traffic intersection, and the FERMI facility of Elettra Sincrotrone. The goal was to provide practical examples of how RL can be used in facing real-world control problems, leading to effective control policies, thus positively answer the first research question **(Q1):** of Chapter 1.

In the former example, we modeled the decision process of a traffic intersection controller as a DDMDP, and proposed two different RL-based algorithms to compute the optimal policy to decide whether the traffic light will be green or red in each lane for a certain period of time: a fixed RL timing approach, and an event triggered RL. Numerical results show that both approaches fit the problem well, thus leading to a policy that reduces queues and vehicle waiting times. In particular, the proposed event-driven solution seems to lead to a better traffic management than those obtained by a fixed timing RL and a simple (no-smart) square wave traffic light. Future investigations may include the HDVs dynamics modeling in the learning procedure, and also additional analysis of the effectiveness as the hyperparameters change (e.g., the $T_R L$, d_a, T_{delay}, T_{alert}). Given the complexity of the considered environment, tests have been performed only in simulation, despite dealing with a problem that comes from real-world.

In the latter application, RL has been used to face, respectively, the attainment of the optimal working point, and its recovery after a machine drift on the FERMI facility. We approached the two tasks using an episodic Q-learning with linear function approximation, and a non-episodic NPG REINFORCE, respectively. In this case as well, the experimental results highlight the effectiveness of the RL approaches. Based on the promising results, future works may include deep Q-learning solutions, thus allowing to work directly with the images detected by the sensors.

In the following chapter, we move to the second problem faced in this dissertation: the reality gap of RL robot controllers (c.f., (**Q2):** in Chapter 1).

4 RL for robot control

The growing demand for robots able to act autonomously in complex scenarios has widely accelerated the introduction of Reinforcement Learning (RL) in robots control applications.

In some well-known RL tasks, such as pole-balancing, grid-search, or mountain car, state and action spaces of the system are small enough to allow approximating policies through tables [158], i.e., actions and states can be dealt with as finite discrete variables. However, the higher the complexity of the system to control, the more ineffective the tabular approaches become [158]. Indeed an increase in system complexity is often related to an increase of the state and action spaces dimensions, which makes a tabular approach intractable. In challenging cases, such as robot control, treating state and action as continuous variables in a compact set is a more appropriate way to deal with the problem [93, 154]. For this purpose, approximators of the policy or of some supporting element, such as value function, or of both, are required [158]. When Deep Neural Networks (DNNs) are employed as approximators, the approach is referred to as Deep Reinforcement Learning (DRL); it allows to develop RL controllers with less manual feature engineering than classic tools (radial basis functions, tile coding, etc.) [15, 94, 122]. On the other hand, when DRL is directly employed on the robot in real-time, it results in considerably long training times. Moreover, due to the intrinsic trial and error nature of RL, a real-world training, in particular during the exploration phase of the state and action space, can lead to unsafe actions of the robot. Therefore, a way to train robots safely and quickly is needed.

Simulators allow to easily address these problems once they are provided with a model of the robot dynamics able to replicate the actual behavior as closely as possible. In principle, simulators allow to train the controller with faster and safer procedures: once the policy has been learned, it is transferred to the real system (*sim-to-real* transfer) [25]. However, sim-to-real transfer is only effective when the simulator is given a sufficiently accurate model of the real robot and the environment [147]; unfortunately, the more accurate the simulation, the heavier the computational cost. A less accurate simulator is therefore often preferred, although it may result in a less effective sim-to-real transfer. The phenomenon in which a controller learned on simulator degrades once applied on the real world is the so-called *reality gap* (RG) [82]. In the worst case, the RG leads to a failure of the policy when applied on the real world, which means a robot unable to achieve its goal.

RL is not the only approach that can be affected by the RG. Any technique in which the controller design relies on a simulator of the real system can potentially exhibit a reality gap [77]. Indeed, several works faced the RG problem in other frameworks, such as Evolutionary Computation [82, 138, 125, 61, 72, 116, 95, 96] or Model Predictive Control [18, 90, 153, 127].

However, here we focus our attention only on those works facing the RG problem on robot controllers learned with RL. Most of the many solutions proposed in the literature, are task-dependent and/or have been tested on a specific task only. The outcomes are that: (a) generalization is not ensured, (b) and a comparison between different approaches is not feasible.

Although a sketch of the current state of the art is already proposed in [191], here we conduct a more in-depth analysis. We introduce the main concept behind the RG in a general RL framework (Section 4.1) and we survey relevant and recent literature concerning RG in the context of robot control with RL (Section 4.2). According to our analysis, the approaches for coping with RG in this context fall into three broad categories: domain randomization (DR), adversarial reinforcement learning (ARL), and transfer learning (TL). We use the framework presented in Chapter 2 to explain in detail the generic behavior of each of the above approaches. We aim (1) to provide a systematic picture of the literature concerning how to solve the RG problem in robot control tasks with RL; (2) to clarify the differences between the three main identified approaches by highlighting the relative pros and cons; and (3) to identify new possible research areas.

For the purpose of evaluating and comparing RL approaches in robotics, in Section 4.3 we propose a novel index that measure the proneness to exhibit the RG, besides the usual indexes for measuring effectiveness and efficiency.

Finally in Section 4.4 we provide a practical example of the reality gap problem on a 6DoF robotic arm, also characterizing some possible modeling errors affecting controller performances in a sim to real transfer.

4.1 RL in robotics and the reality gap

The motivation for using RL in robotics is to make a robot autonomous in finding an optimal policy, through *trial and error* interactions with its environment, without an explicit knowledge of the model (model-free). However, the most effective methods to date are model-based [16, 1, 51]. In addition, policy search approaches result in more efficient trainings in terms of time needed for convergence [68, 111, 162, 135, 136, 33]. Regardless of the specific RL method being used, the application of RL enabled researchers and practitioners to face different significant robotic tasks (e.g., manipulation, navigation, motion control, etc.). In

Table 4.1: Overview of the robotic tasks involving RL controllers.

Tasks	References
Navigation	[146], [89], [79], [192]
Manipulation	[145], [134], [108], [84], [188], [154], [147], [103]
Motion control	[36], [107], [137], [184], [120], [160], [65], [170], [129], [131], [12]
Locomotion	[36], [107], [137], [70], [155], [100]

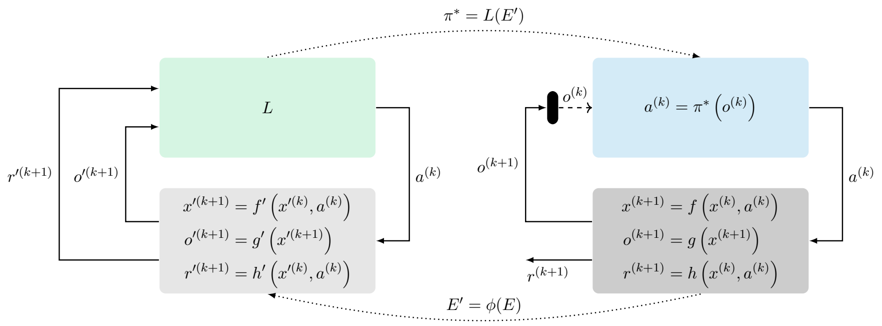

$\pi^* = L(E')$
L
$a^{(k)} = \pi^*\left(o^{(k)}\right)$
$o^{(k)}$
$r'^{(k+1)}$
$o'^{(k+1)}$
$a^{(k)}$
$o^{(k+1)}$
$a^{(k)}$
$x'^{(k+1)} = f'\left(x'^{(k)}, a^{(k)}\right)$
$o'^{(k+1)} = g'\left(x'^{(k+1)}\right)$
$r'^{(k+1)} = h'\left(x'^{(k)}, a^{(k)}\right)$
$x^{(k+1)} = f\left(x^{(k)}, a^{(k)}\right)$
$o^{(k+1)} = g\left(x^{(k+1)}\right)$
$r^{(k+1)} = h\left(x^{(k)}, a^{(k)}\right)$
$r^{(k+1)}$
$E' = \phi(E)$

Figure 4.1: Schematic representation of the application of RL to the robot control problem using simulation. First, given the real robot and it environment E (block at bottom right), a simulator E' (block at bottom left) is obtained by modeling E using ϕ. Then an agent L (block at top left) learns a policy π^* in simulation. Finally, the learned policy is transferred to the real robot where it can be deployed (block at top right).

Table 4.1, we report the tasks that are more often considered[1], along with a few significant research papers, some of which (those dealing with sim-to-real transferability) are surveyed in the present study.

[1]The main tasks addressed in the literature are: navigation, manipulation, locomotion, and motion control. The former concerns the ability of robots to determine their position in a given reference frame, and plan a path leading them to some target locations. Manipulation refers to the set of tasks in which robots interact with objects around them, e.g., grabbing an object, opening a door, packing an order in a box. Locomotion encompasses all the various applications in which robots have to transport themselves from one place to another. Finally, motion control is the ability of the robot to determine a temporal sequence of control inputs to achieve a desired movement. Here we include in this latter subcategory all motion tasks not included in the previous categories.

When tackling a robot control problem with RL, according to the formulation and notation provided above, the designer should ensure that (a) the environment E captures the robot-surrounding environment dynamics; (b) the observations $o \in O$ include proprioceptive measures useful to capture the robot dynamic evolution (e.g., joints position and velocity); (c) the control inputs $a \in A$ correspond to values for the appropriate robot actuators (e.g., torques to be applied to joints or desired acceleration/speed/position, depending on the specific control system).

A first issue concerns the representation of $V_\pi(x)$, and $Q_\pi(x, a)$. While in some simple RL scenarios O and A sets can be discretised over a finite range, hence allowing a tabular representation of $V_\pi(x)$ or $Q_\pi(x, a)$, in robot control problems the physical nature of observations and control inputs advocates for a finely discretised (ideally, continuous) representation of O and A. In this case, table may require a huge amount of memory and the problem gets practically intractable. An alternative and often suitable solution to escape tabular representation is function approximation, recently addressed by DRL. Here DNNs are used as function approximators of $V_\pi(x)$, $Q_\pi(x, a)$, or directly π, and a loss function η_θ is designed in order to guide the training of the network itself. For a more detailed description of DRL, including recent efforts and some applications, we refer the reader to [15, 117].

A second issue, pivotal for the aim of the present work, derives from the trial and error process employed by RL and its direct application to the robot: learning time may become too long, thus unpractical, and the risk of damaging the robot, or more in general the environment, may be too high. Typically, simulators are used for addressing this issue; i.e., an *environment mapping operator* ϕ is assumed to exist (albeit unknown in practice) such that $E' = \phi(E) = (X', A', O', f', g', h')$ is a digital approximated copy of E. Intuitively, ϕ corresponds to modeling a real system described by E as a simulated system described by E' such that a policy learned on E' can be applied to E. The resulting controller design and test are, therefore, split in two distinct phases that are performed on two different environments: (i) an agent L interacts with a simulated environment $E' = \phi(E)$ and outputs a controller π^*; (ii) the resulting π^* is applied on the real E. In this scenario, outlined in Figure 4.1, the latter step, that in general can be defined as E'-to-E transfer, can be renamed as *sim-to-real* transfer, given the nature of the considered E' and E.

As will be clear from the reviewed examples in the following sections, a controller learned in simulation often exhibits performance losses when applied on the real robot and, in the worst scenario, totally fails the task. From the point of view of the designer, this issue becomes relevant when, although a safer and faster training and an effective test have been carried out on E', the policy learned in simulation does not achieve the goal in real world. In such a situation, the learning algorithm is affected by an intolerable RG and the corresponding learned policy π^* is said to be non-transferable.

Note that, although it is not always clearly emphasized, a mandatory step ahead of the sim-to-real transfer is to perform a test of the learned policy on the simulator itself[2]. Otherwise, there is no guarantee of the learned controller effectiveness in achieving the task even in simulation. For instance, a typical learning stop criterion consists in terminating the training when the moving average of the cumulative reward settles down. However, this empirical rule does not ensure that the learned controller is able to correctly perform the task. The learning algorithm may have been trapped in a local maximum, and the resulting controller may have a completely unexpected behavior in tests, even on the simulator.

If the test on the simulator is effective, possible reasons for a performance loss in a test on the real robot, and therefore for RG, are:

(a) ϕ operator is unrealistic: E and E' differences in f,f' and/or g, g' are such that, applying the same input on both environments, the

[2]This only applies to those situations where the controller is transferred on the real robot. As shown in Section 4.2.3, in some scenarios, the training continues on the real robot. Hence a test procedure is not needed in these cases.

resulting o and o' are different; or, possibly, E and E' differences in h and h' lead to two different optimization problems $J_\pi(x) \neq J_\pi(x')$;

(b) L is unable to output a π^* sufficiently robust to possibly small and unavoidable errors in ϕ.

However, the particular case in which h' is very different from h is unusual, in practice. Typically, h is properly designed and remains "the same" for both E and E' (for this reason hereinafter we consider $h' = h$). Therefore, the RG can be essentially attributed to a mismatch between E and E' and the possible solutions may be: (a) improve E', by properly adjusting ϕ (not always possible because it could result in an excessive computational effort or because of lack of knowledge); (b) make the controller more robust to model errors.

Note that, in general, it is not required that the controller behaves identically when applied to E and E', but, rather, that it is E'-to-E transferable, i.e., sim-to-real transferable. In practice, this requirement translates to a (subjectively) properly bounded RG. Clearly, if an appropriate behavior is only reached when E' is an identical copy of the robot E ($E' = \phi(E) = E$), using E' rather than E has no benefit in reducing the overall learning time. But still, E' may be useful for addressing risks concerning safety.

Figure 4.2 summarizes a generic routine for investigating the presence of RG. We start by (1) performing a training on E'; (2) then we test the resulting policy π on the same E', to check its effectiveness in reaching the task, and, only if the test ends successfully, (3) we test π also on E; otherwise, we revert to (1). Once that the (3) has been executed, (4) we compare the performance obtained in the π tests on E and E'.

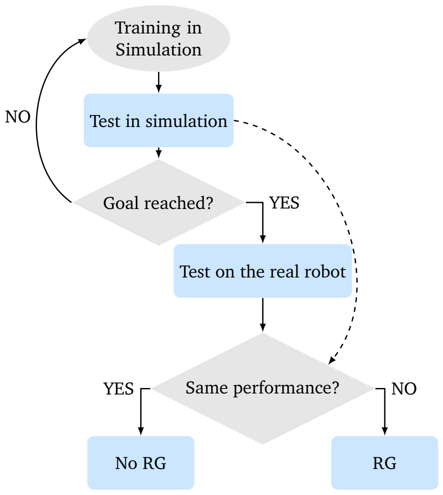


Figure 4.2: Flowchart of the routine to test the presence of the RG.

4.2 Methodologies for solving the RG

Here, we focus only on those articles that meet all the following requirements: (i) address explicitly the RG problem, (ii) deal with robot control applications, and (iii) employ RL techniques. We have identified three major categories of approaches for addressing the RG in this scenario: domain randomization (DR), adversarial RL (ARL), transfer learning (TL). All the articles discussed below are summarized in Table 4.2 according to this categorization. The table also shows, for each article, if the authors conducted experiments only in simulation (sim-to-sim) or (also) on real robot (sim-to-real), and specifies the employed simulators.

Due to task diversity (Figure 4.3 shows a visual summary of the robotic tasks) and the lack of a common theoretical framework for the RG, the surveyed articles do not present their results in a way that permit a systematic comparison. However, we provide a general formal definition for each of the previously mentioned categories, according to the formalism of Chapter 2, which allows to understand each approach.

Table 4.2: List of the surveyed articles, specifying the category (Cat. column). S2S and S2R columns show whether in the article the proposed technique was evaluated by performing sim-to-sim or sim-to-real experiments, respectively.

Ref.	Cat.	S2S	S2R	Simulator
[36]	TL	✓	✓	MuJoCo [165]
[146]	DR	✓	✓	Blender [24]
[145]	TL		✓	MuJoCo
[107]	DR	✓		MuJoCo
[137]	ARL	✓		MuJoCo
[70]	TL	✓	✓	SimSpark [157], Gazebo [128]
[184]	TL	✓		MuJoCo, DART [99]
[120]	DR	✓		Vortex [40], Bullet[41]
[134]	DR		✓	MuJoCo
[108]	DR		✓	PyBullet [42]
[160]	DR	✓	✓	PyBullet
[65]	TL	✓	✓	MuJoCo, PyBullet
[170]	DR		✓	MuJoCo
[89]	TL		✓	Gibson [186]
[129]	DR	✓	✓	MuJoCo
[131]	ARL	✓		TORCS [185]
[84]	DR		✓	PyBullet
[188]	TL		✓	Gazebo
[155]	DR	✓	✓	MuJoCo
[79]	DR	✓	✓	Gazebo

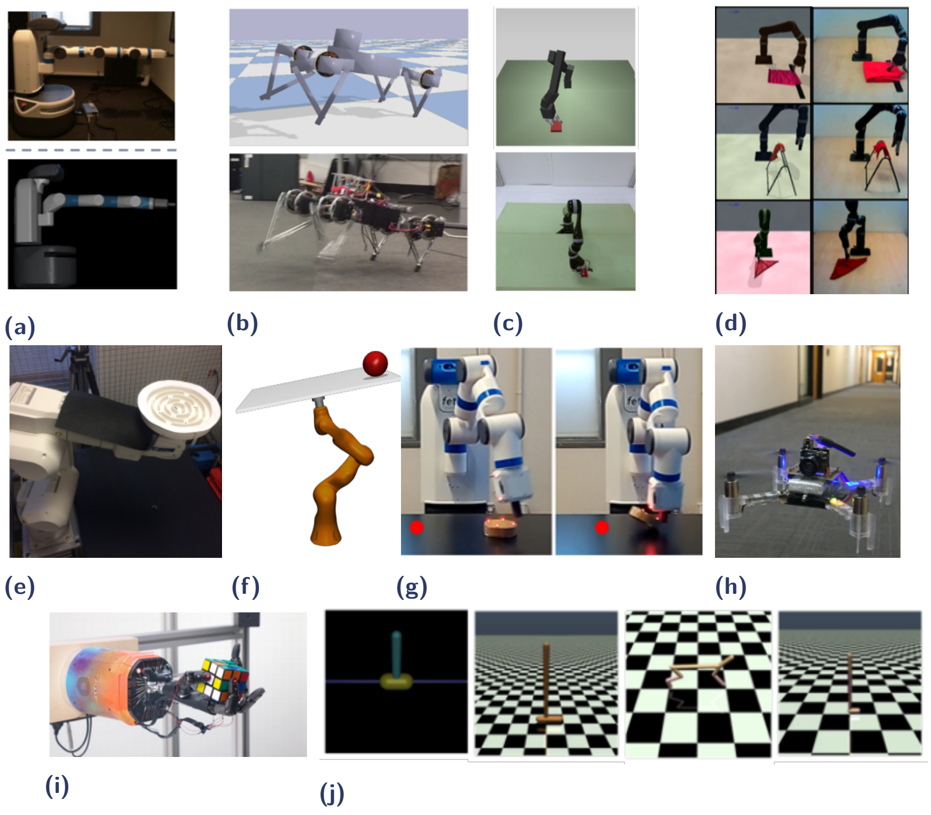


Figure 4.3: Overview of some robotic tasks considered in the surveyed articles to address the RG. (a) The Fetch robot used in [36]. (b) The Minitaur of [160]. (c) The robot employed for manipulation task in [144]. (d) The robotic arm engaged in deformable object manipulation of [108]. (e) The Marble maze game of [170]. (f) The ball on plate system used by [120]. (g) The Fetch robot used for the pushing task of [134]. (h) The quadrotor employed for the autonomous navigation task of [89]. (i) The five-finger humanoid hand used in [129]. (j) The classical Open AI Gym environment used to test in simulation several strategies [36, 107, 137, 184].

4.2.1 Domain randomization

Domain randomization (DR) has already achieved good results in sim-to-real transfer of robotics controllers outside RL [164, 83, 115, 179, 173, 160]. The main idea behind this approach is what in control theory is called *robust control under either parametric or non parametric uncertainty* [23, 5], that is the design of controllers able to guarantee certain properties despite some tolerable parameters variations and/or noise.

We call $\tilde{E}' = \tilde{\phi}(E) = (\tilde{X}', \tilde{A}', \tilde{O}', Z', \Upsilon', \tilde{f}'_{\xi'}, \tilde{g}'_{\psi'}, h)$ a corrupted simulator described in which Z' is the process disturbances set, Υ' is the measurement disturbances set, $f'_{\xi'} : \tilde{X}' \times \tilde{A}' \times Z' \to \tilde{X}'$ the corrupted and parametric transition function, with parameters $\xi' \in \Xi'$, and $g'_{\psi'} : \tilde{X}' \to \tilde{O}'$ the corrupted and parametric observation function, with parameters $\psi' \in \Psi'$.

Given a parametrisation ξ', ψ', starting from an initial state $\tilde{x}'^{(0)}$ and subject to a control sequence $\tilde{a}'^{(0)}, \tilde{a}'^{(1)}, \dots$, a process disturbance sequence $\zeta'^{(0)}, \zeta'^{(1)}, \dots$, and a measurement disturbance sequence $\upsilon'^{(0)}, \upsilon'^{(1)}, \dots$, a corrupted simulator $\tilde{E}'$ evolves according to:

$$\tilde{x}'^{(k+1)} = \tilde{f}'_{\xi'}\left(\tilde{x}'^{(k)}, \tilde{a}'^{(k)}, \zeta'^{(k)} | \xi'\right) \tag{4.1}$$

$$\tilde{o}'^{(k+1)} = \tilde{g}'_{\psi'}\left(\tilde{x}'^{(k+1)}, \upsilon'^{(k)} | \psi'\right) \tag{4.2}$$

$$\tilde{r}'^{(k+1)} = h\left(\tilde{x}'^{(k)}, \tilde{a}'^{(k)}\right). \tag{4.3}$$

The main idea behind DR is that, during training, L selects ξ' and ψ', interacts with the resulting environment $\tilde{E}'$, and updates a controller π by observing the consequences (in terms of reward) of selected control inputs $\tilde{a}'^{(k)}$, process disturbances $\zeta'^{(k)}$, and measurement disturbances $\upsilon'^{(k)}$. Its final goal is twofold: (a) maximize the finite horizon discounted reward in a perturbed environment (see Equation (2.4)) and (b) find a solution π^* which ensures a loss in performance lower than a threshold when applied on different domains of the same distribution. In particular, the final controller π^* sim-to-real transferability is here seen as a form of controller robustness obtained by training π in a *collection of environment models*, chosen by L, instead of a single one—Figure 4.4 graphically summarizes this process. The resulting controller π^*, learned by maximizing the finite horizon discounted reward $J_{\pi,T}$ under these conditions, is expected to be robust to perturbations. Therefore, if these perturbations are such that the $L \leftrightarrow E$ interaction returns

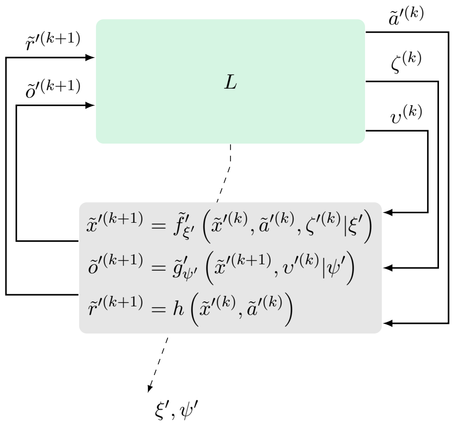

$\tilde{a}'^{(k)}$
$\tilde{r}'^{(k+1)}$
$\zeta^{(k)}$
$\tilde{o}'^{(k+1)}$
L
$\upsilon^{(k)}$
$\tilde{x}'^{(k+1)} = \tilde{f}'_{\xi'}\left(\tilde{x}'^{(k)}, \tilde{a}'^{(k)}, \zeta'^{(k)} | \xi'\right)$
$\tilde{o}'^{(k+1)} = \tilde{g}'_{\psi'}\left(\tilde{x}'^{(k+1)}, \upsilon'^{(k)} | \psi'\right)$
$\tilde{r}'^{(k+1)} = h\left(\tilde{x}'^{(k)}, \tilde{a}'^{(k)}\right)$
ξ', ψ'

Figure 4.4: Schematic representation of $L \leftrightarrow \tilde{E}'$ interaction in DR approaches.

a π^* affected by a tolerable RG, the result is a sim-to-real transferable controller.

Table 4.3 summarizes the articles that tackle the RG using the DR approach. The table shows also the employed learning algorithms and the considered tasks.

We remark that in some of these studies the actual sim-to-real transferability is not evaluated (see Table 4.2); instead the controller robustness with respect to the perturbations is tested. We discuss each of the paper below.

In Sadeghi *et al.* [146] authors train a vision-based navigation policy entirely in simulation, trying to use it on a real quadrotor without performing additional real training runs. During training, at each time k, the state of the system is here represented by an indoor synthetic image $I^{(k)}$ generated by a renderer. Images are generated in order to reproduce different hallways and a variety of environment parametric settings (ξ',ψ'). First, a Deep Convolutional Neural Network is learned in order to predict the collision probability for each $I^{(k)}, a^{(k)}$. Then, a Deep RL agent is trained for fine-tuning the previous model to provide the action-value function $Q(I^{(k)}, a^{(k)})$. Hallway randomization enacts a wide variety of environments, and shows very good performance during

Table 4.3: List of the surveyed articles that apply the DR approach.

Ref.	Algorithm	Tasks
[146]	DQN^h	Vision-based flight
[107]	TRPO^a	Inverted pendulum, Half cheetah [26], Hopper [26], Walker [26]
[120]	TRPO^a	Ball on plate with robotic arm
[134]	HER^b+RDPG^c	Pushing task with robotic arm
[108]	DDPGfD^f	Deformable object manipulation
[160]	PPO^e	Trotting and galloping of quadruped
[170]	A3C^d	Marble maze game with robotic arm
[129]	PPO^e	Rubik's cube with robotic hand
[84]	QT-Opt^g	Rvision-based control task of a robotic arm for grasping
[155]	PPO	Climb and descend stairs with bipedal robot
[79]	A3C	Wheeled mobile platform navigation

^{a}Trust Region Policy Optimization [150], ^{b}Hindsight Experience Replay [11], ^{c}Recurrent Deterministic Policy Gradient [73], ^{d}Asynchronous Actor-Critic Agents [112], ^{e}Proximal Policy Optimization [151], ^{f}Deep Deterministic Policy Gradient from Demonstration [175]. ^{g}Q-function Targets via Optimization [88] ^{h}Deep Q Network [113]

test, both in simulation and on the real world, even with environments never seen during training. However, performance falls when the drone encounters reflective glass doors, thus resulting in a crash.

Mandlekar *et al.* [107] introduces an algorithm, called Adversarially Robust Policy Learning (ARPL), to teach a controller to correctly behave in presence of increasing adversarial perturbations. The agent uses a curriculum learning approach [20], in which ξ', $\upsilon'^{(k)}$, and $\zeta'^{(k)}$ alternately assume the form of isometrically scaled versions of Fast Gradient Sign Method (FGSM) [67]. Here, the controller is parametrized by θ (π_θ) and updated following the on-policy vanilla Trust Region Policy Optimization (TRPO) [150]. The key idea is to use a corrupted simulator $\tilde{E}'$ in training, and then testing the resulting π^* on a different corrupted simulator $\tilde{E}'' = (X'', A'', O'', Z'', \Upsilon'', f'_{\xi''}, g'_{\psi''}, h)$ environment, obtained with different perturbations $Z'' \neq Z'$, $\Upsilon'' \neq \Upsilon'$, $\Xi'' \neq \Xi'$ and $\Psi'' \neq \Psi'$. These perturbations are such that π^* is misled to provide wrong control inputs $a''^{(k+1)}$. The choice of adversarial perturbations is motivated by the fact that by employing them, the resulting models are likely to generalize well [159]. The ARPL algorithm has been tested in several benchmark examples (Inverted pendulum, Half cheetah, Hopper, Walker) and seems to deliver promising results, exhibiting significant robustness. However, examples of sim-to-real controller transferability have not yet been provided.

The Simulation-based Policy Optimization with Transferability Assessment (SPOTA) algorithm, designed in [120], uses randomized physics parameters, drawn from a probability distribution parametrized by κ, $\xi' \sim \rho_\kappa(\xi')$, to perform a robust optimization of the controller. In SPOTA, the controller is trained on model ensembles, according to the following 4 phases: (i) learn a candidate solution π_θ^C using a TRPO updating rule; (ii) learn n_R reference solutions $\pi_\theta^{R_j}, j = 1, \ldots, n_R$ on n_R different $\tilde{E}'$, each obtained for different ξ' and ψ' settings; (iii) compare the performance of candidate C with that of each reference R_j in the same condition of R_j; and, finally, (iv) decide whether or not stop the learning. The last step is carried out by introducing a Simulation Optimization Bias (SOB) concept: an error caused by an optimistic bias of the optimization procedure, whose existence has been proven by [76]. The authors have assumed that it can be treated as the error between the finite horizon discounted reward $J_{\pi_\theta^{*R},T}$ obtained by considering the reference solutions and the finite horizon discounted reward $J_{\pi_\theta^C,T}$ of candidate solution. Taking into account that an RL approach in a stochastic setting allows to find only estimates of $J_{\pi_\theta^{*R},T}$ and $J_{\pi_\theta^C,T}$, the authors have derived an upper bound for the tolerated SOB of the candidate solution, called Upper Confidence bound on Simulation Optimization Bias (UCSOB). In order to ensure a desired performance β, the final candidate solution UCSOB must be lower than β. In this framework, the authors have tested the algorithm by developing a controller for a ball on plane task, governed by a robotic arm, in the same simulator (obtaining satisfactory results) and varying the physics engine (with worse outcomes). However, although in this case a sim-to-sim transferability test has been carried out, a sim-to-real controller test has not been done.

Peng, Andrychowicz, *et al.* [134] shows the effectiveness of memory-based policies (i.e., policies learned by using past memory for future learning [114, 35, 133]) to deal with the RG, introducing DR to generalize environment dynamics. Hindsight Experience Replay [11] has been used for the purpose: a technique able to generalize over different goals using past experience as a baseline. In this case, the parameters ξ', the measurement noise $v'^{(k)}$, and the time step Δt are sampled according to a distribution, which is a design parameter. In particular, ξ' is kept locked for an entire episode, while the remainder are varied at each time step. The proposed solution, learned using a RDPG algorithm (off-policy), has been tested on a robotic pushing task and, when transferred to reality, shows performances comparable to those obtained in simulation, despite poor calibration.

An improved version of DDPG [175] is adopted in [108] to solve deformable object manipulation tasks in simulation. The resulting controller transferability on the real robot is therefore tested. In particular, a robotic arm is involved in three different towel folding tasks, in which RGB images are included in the observation o. The DR is here implemented by sampling some environment values from either normal or uniform distributions around noisy ground truth estimates. Experimental results suggest that randomization of extrinsic camera parameters (i.e., position and orientation) is particularly useful for sim-to-real transfer, since the controller has an evident sensitivity to changes of its position. Besides, they show that heavy randomization can lead to unsuccessful transfers.

Controller sim-to-real transferability has also been tested on locomotion tasks of a Minitaur quadruped of Ghost Robotics [160]. Here authors have used Proximal Policy Optimization (PPO) to learn π^* and have observed the impact of two different solutions to reduce the RG: (a) improving simulated model via system identification; (b) using randomized $\zeta'^{(k)}$, $\upsilon'^{(k)}$, and ξ' to learn robust controllers as the observation space changes. Obtained results suggest that simulators improvement is an essential requirement since, as the model becomes less adequate, not even a robust controller is able to avoid a large RG. The authors of the cited paper also pointed out that considering a large observation space does not always bring benefits. On the contrary, their evaluations have showed that controllers learned in simulation with large observation space lead to bad results when transferred to real robot.

Van Baar *et al.* [170] shows the benefits of using DR and Asynchronous Actor Critic Agent (A3C) algorithm [112] (on-policy) for learning the controller, with respect to not using DR. The parameters ξ' are here randomly sampled according to a uniform distribution. Both controllers are then applied on real-world robot and the fine-tuning time required to convergence is compared. The analyzed task is a Marble maze game driven by a robotic arm and the results show that there is a trade-off between controller robustness and fine-tuning steps. Controller learned through DR requires fewer fine-tuning steps than the remaining one, further proof of an existing trade-off between efficiency and RG.

In a quite recent work [129], Automatic Domain Randomization (ADR) is proposed in order to transfer a policy learned in simulation on the real system, framing it in a manipulation task of Rubik's cube with a robotic hand. Here, the RL agent does not solve Rubik's cube but “only” learns, using a PPO algorithm (on-policy), how to move correctly the robotic hand in order to perform control inputs suggested by another non-AI based algorithm. What changes from standard DR idea are $\rho_\kappa(\xi')$ and $\rho_\kappa(\upsilon^{(k)})$ distributions (parametrized by κ) that allow to randomly select $\upsilon^{(k)}$ and ξ'. Indeed, while in other DR approaches these distributions are parametrized with fixed κ, in ADR κ changes during learning procedure. In particular, these additional environments, obtained with different

κ, are added to the considered collection of environment models only when a lower performance limit is reached (i.e., a fixed number of successful episodes are performed). The developed controller has been firstly tested in environments in which distributions were manually tuned, achieving good results. In addition, a sim-to-real transfer is performed, with worse results.

A Randomized-to-Canonical Adaptation Network (RCAN) is conversely introduced in [84]. The main idea is to map the observations collected on the simulated domain as well as those collected on the real domain into a common further domain called the *canonical domain*. The approach has been applied to a vision-based robot grasping task, and the canonical domain consists of extremely simplified images whose purpose is capturing just the relevant information for the task. The map is learned by using an image-conditioned Generative Adversarial Network (cGAN) [81], able to map an image of a domain D into an adapted image of the canonical domain D_c; i.e., $G : D \to D_c$. The resulting image of D_c is then sent to the controller which is learned by using Q-function Targets via Optimization (QT-Opt) [88]. During training, cGAN receives randomized simulated images, sampled from the trajectories, and learns to convert them in canonical images. The resulting observations are then used by the QT-Opt to produce the policy. In the test procedure, the real-world images are mapped into canonical images and sent to the controller. The proposed approach returns excellent results, however, an effective transfer is not always achieved. In particular, when the cGAN during training is fed with images sampled only from non-successful trajectories, the final controller results in an unsatisfactory transfer.

Siekmann *et al.* [155] proposes a simple terrain randomization to learn robust proprioceptive controllers for bipedal robots involved in the task of climbing and descending stairs. They model the policy with a Long short-term memory (LSTM) network, for its capability of processing temporal sequences. Indeed, unlike feed-forward neural networks, LSTMs are equipped with a feedback mechanism that allows them to process sequences of input data, without treating each sample of the sequence independently. They retain useful information about earlier data points in the sequence, aiding in the processing of new data points. The authors compare the performance of three different controllers π: (i) A learned LSTM controller with different terrain parameters ψ, (ii) a feed-forward NN controller learned with different ψ' terrain parameters, and (iii) a LSTM controller learned on a single simulated environment. The experimental results show that the first π is the one with the highest overall probability of success in the task. Thus, the combination of LSTM and DR seems to be an effective solution to the problem for the tested task.

Finally, Hu *et al.* [79] face the reality gap of a controller involved in a wheeled robot navigation task. The proposed solution tries to render the controller robust to possible parametric errors in the model, but also to possible disturbances that corrupt its dynamics. To this end (i) a terrain randomization is performed by varying its viscosity and inclination ψ', (ii) disturbances on the travel distance and on the yaw rotation $\upsilon_1^{(k)}$ are randomly imposed to the dynamics, (iii) latency disturbances are imposed at each time-step, hypothesizing that a latency between perception and movement performance occurs $\upsilon_2^{(k)}$, and (iv) the pose-estimation error $\upsilon_3^{(k)}$ is also taken into account. The resulting π, learned entirely in simulation, results in an effective application on the real-world environment. Moreover, a comparison with some state of the art solutions for robot navigation highlights the better performance of the proposed approach in terms of success rate as well as cumulative travel distance, and time required for task execution.

4.2.2 Adversarial RL

In the adversarial RL (ARL), the agent L is composed of two sub-agents: the *protagonist* L_P and the *antagonist* L_A. The underlying idea resembles the one behind domain randomization: enforce robustness (and, hence, improve controller transferability) by training the controller in a collection of environment models instead of a single one. In the case of ARL, however, the diversity is obtained by training a secondary

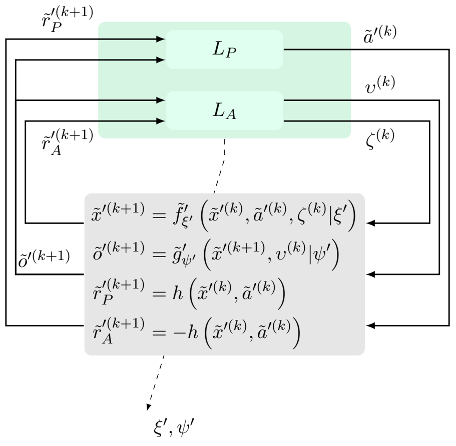

$\tilde{r}_P'^{(k+1)}$
L_P
$\tilde{a}'^{(k)}$
$\upsilon^{(k)}$
L_A
$\zeta^{(k)}$
$\tilde{r}_A'^{(k+1)}$
$\tilde{x}'^{(k+1)} = \tilde{f}'_{\xi'}\left(\tilde{x}'^{(k)}, \tilde{a}'^{(k)}, \zeta^{(k)} | \xi'\right)$
$\tilde{o}'^{(k+1)}$
$\tilde{o}'^{(k+1)} = \tilde{g}'_{\psi'}\left(\tilde{x}'^{(k+1)}, \upsilon^{(k)} | \psi'\right)$
$\tilde{r}_P'^{(k+1)} = h\left(\tilde{x}'^{(k)}, \tilde{a}'^{(k)}\right)$
$\tilde{r}_A'^{(k+1)} = -h\left(\tilde{x}'^{(k)}, \tilde{a}'^{(k)}\right)$
ξ', ψ'

Figure 4.5: Schematic representation of $L \leftrightarrow \tilde{E}'$ interaction in ARL approaches.

controller (the adversarial) to generate more difficult models to handle (those that minimize the cumulative reward). Figure 4.5 graphically summarizes the process of ARL.

Given $\tilde{E}' = (\tilde{X}', \tilde{A}', \tilde{O}', Z', \Upsilon', \tilde{f}'_{\xi'}, \tilde{g}'_{\psi'}, h)$ a corrupted simulator of E, evolving according to Equations (4.1) to (4.3), L_P and L_A interact with $\tilde{E}'$ seeking to maximize their respective discounted cumulative reward [169].

We denote with $\tilde{r}_P'^{(k+1)}$ the reward of L_P. A common choice [101] is to provide L_A with a reward $\tilde{r}_A'^{(k+1)} = -\tilde{r}_P'^{(k+1)}$. As a result two controllers are learned:

- π_P, whose target is to maximize the cumulative reward over time, resulting in the final controller which will be tested in a $\tilde{E}'$-to-E transfer;
- $\pi_A : \tilde{O}' \to Z' \times \Upsilon' \times \Xi' \times \Psi'$ that searches for those environment perturbations or parameters variations that *minimize* the same cumulative reward over time.

The outputs of L_A and π_A are perturbations and parameters variations of $\tilde{E}'$. In a noise-corrupted simulated environment, the observed reward and hence the discounted reward will depend on the disturbances as well as on the policy. Since, in ARL, disturbances are generated by the adversarial agent, the finite horizon discounted reward will depend on both policies. To catch this dependency we can write $J_{\pi_P,\pi_A}\left(\tilde{x}'^{(0)}\right)$. The resulting L goal can be compactly stated as:

$$\max_{\pi_P} \min_{\pi_A} J_{\pi_P,\pi_A}\left(\tilde{x}'^{(0)}\right), \tag{4.4}$$

falling in a *worst-case* approach that in control theory is known as *min-max optimal control* (also referred to H_∞-control) [105, 104, 47].

Consequently, in ARL, $\tilde{E}'$ interacts simultaneously with L_A and L_P, thus $L \leftrightarrow \tilde{E}'$ results in $L_A \leftrightarrow \tilde{E}' \leftrightarrow L_P$ and the global L task is to find:

$$\pi_L^* = \arg\max_{\pi_P} \arg\min_{\pi_A} J_{\pi_P,\pi_A}. \tag{4.5}$$

Table 4.4 summarizes the articles that use ARL for learning robust controllers, along with the respective learning algorithms and the considered tasks.

Table 4.4: List of the surveyed articles that apply the ARL approach.

Ref.	Algorithm	Tasks
[137]	TRPO[a]	Inverted pendulum, Half cheetah, Swimmer [26], Hopper, Walker 2D, Ant [26]
[131]	Ensemble DQN[b]	Autonomous driving

[a]Trust Region Policy Optimization [150], [b]Ensemble Deep-Q Network [130].

The ARL approach was firstly introduced by [137], with the Robust Adversarial Reinforcement Learning (RARL) algorithm. There, π_P (the protagonist's policy) is trained to work in presence of an adversary (π_A), able to inject destabilizing disturbances to environment (in particular only ξ' disturbances). The proposed solution can be summarized in two main steps that are repeated n_{iter} times: (i) learn the protagonist policy π_P while keeping the adversary one fixed; (ii) learn the adversary policy π_A while keeping π_P fixed. The experiments have been done on several OpenAI Gym environments [26]. In a first experiment, the authors compare mean and variance of the cumulative reward over 50 RARL policies, obtained using different seeds and initialization, with TRPO ones. For all tasks RARL behaves better than TRPO in terms of mean and variance. In a second experiment, [137] shows that RARL behaves better than TRPO under adversarial attacks while keeping hold the protagonist. Finally, in a third experiment, the authors introduce different $\zeta^{(k)}$ in the test phase, and again obtain better results with respect to TRPO.

Pan *et al.* [131] builds on the RARL idea by introducing the Risk-Averse Robust Adversarial RL (RARARL) concept: a RARL algorithm in which the protagonist is trained to be risk-averse and the adversarial, in contrast, risk-seeking. The authors of the cited study state that "a robust policy should not only maximize long-term expected reward, but should also select actions with low variance of that expected reward". For that purpose, they train κ different Q-value networks that return κ action-value outputs. The risk of an action is estimated by the empirical variance of these κ Q-values ($\text{Var}_\kappa(Q)$). At the beginning of each training episode one of the κ networks is randomly chosen and employed for control input selection during the entire episode. The protagonist and the antagonist take actions sequentially: the protagonist action-value function Q_{π_P} is augmented by a risk-averse term ($\text{Var}_\kappa(Q_{\pi_P})$), which encourages the choice of lower variance control inputs; the adversary Q_{π_A}, instead, is reduced by a risk-seeking term ($\text{Var}_\kappa(Q_{\pi_A})$) in order to guide it towards higher-variance outputs. The algorithm has been tested in a simulated self-driving task and obtained experimental results highlighting the better robustness of RARARL controller with respect to one subjected to random perturbation in training. During the test, control inputs are selected according to the mean value of the κ networks.

4.2.3 Transfer learning

The previously discussed approaches are aimed at controllers that, once learned in simulation, can be directly transferred on real robots without any (or, at most, with very few) additional training steps. Basically, an agent searches for a sim-to-real transferable controller.

A different perspective is the one adopted in transfer learning (TL) approach. Indeed, its aim is not to find a solution to the RG, but rather to avoid its occurrence by means of two subsequent or simultaneous training phases (first in simulation and second in reality), penalizing the resulting L efficiency.

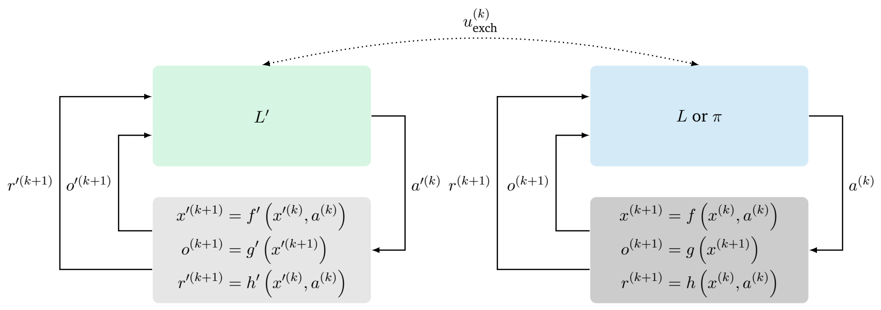
Let us ignore for the moment the RG and suppose to be in a classic RL training scenario, described by $L \leftrightarrow E$. The basic idea of TL is that generalization is possible not only *within* task but also *between* tasks [161].

Definition 5. Given $E = (X, A, O, f, g, h)$ and $E' = (X', A', O', f', g', h')$ two environments, we say that E is *compatible* with E' if and only if any policy for E' is also a policy for E, i.e., $O \subseteq O'$ and $A' \subseteq A$.

Definition 6. The previous two environments are *mutually compatible* if and only if E is *compatible* with E' and E' is *compatible* with E, i.e., $O \equiv O'$ and $A' \equiv A$.

The above definitions ensure that a policy for a system E is also a policy for every system E' that is compatible with E. However, a policy which is optimal for E is in general not optimal for E'.

Therefore, since a task can be entirely defined by an environment, considering a second different but *mutually compatible* environment the controller learned for the first is expected to be a helpful tool to speed-up the second learning process, whether or not it involves the



$u^{(k)}_{\text{exch}}$
L'
L or π
$r'^{(k+1)}$
$o'^{(k+1)}$
$a'^{(k)}$
$r^{(k+1)}$
$o^{(k+1)}$
$a^{(k)}$
$x'^{(k+1)} = f'\left(x'^{(k)}, a^{(k)}\right)$
$o^{(k+1)} = g'\left(x'^{(k+1)}\right)$
$r'^{(k+1)} = h'\left(x'^{(k)}, a^{(k)}\right)$
$x^{(k+1)} = f\left(x^{(k)}, a^{(k)}\right)$
$o^{(k+1)} = g\left(x^{(k+1)}\right)$
$r^{(k+1)} = h\left(x^{(k)}, a^{(k)}\right)$

Figure 4.6: Schematic representation of TL approaches.

same agent. Thereby, in RL, in which the controller is the result of a trial and error process, TL could be employed to speed-up the learning, thus avoiding training from scratch.

Back to the RG, the idea of “recycling" policies between tasks could be useful in speeding the real robot learning procedure or, possibly, while performing a fine tuning of simulated and real agents (L' and L respectively).

In the first scenario, what has been learned in simulation, by using L', is reused (in its entirety or in part) in subsequent phases of real-world training, performed by using L. The expected result is a faster real-world training that bridges the discrepancy between simulator and real robot at its root, i.e., while real agent L is learning its π. Therefore, the transfer occurs once, and only in one direction (from simulation to reality).

In the second case, by providing some real information to simulator and taking example from it, the simulator could more realistically adapt itself to reality, thus reducing performance misalignment and, thereby, the RG. In this situation, the transfer is repeated, and in both directions (sim-to-real and real-to-sim).

Overall, we refer to the exchanged information as $u^{(k)}_{\text{exch}} = \left[u'^{(k)} \; u^{(k)}\right]^T$, in which $u'^{(k)}$ is the sim-to-real transferred information, while $u^{(k)}$ the real-to-sim one (Figure 4.6).

We categorize the different TL approaches based on:

- the kind of information passed through $u^{(k)}_{\text{exch}}$, (e.g., weights of an image processing net, state-control input pairs, policy parameters or even the policy itself);
- the transfer timing of $u^{(k)}_{\text{exch}}$, either continuous (C-TL) or one shot (1-TL), the latter generally occurring at the end of the training in simulation;
- the direction of $u^{(k)}_{\text{exch}}$ transfer, either bidirectional (2D-TL), i.e., $u^{(k)}_{\text{exch}} = [u'^{(k)}\ u^{(k)}]^T$, or unidirectional (1D-TL), i.e., $u^{(k)}_{\text{exch}} = [0\ u^{(k)}]^T$ or $u^{(k)}_{\text{exch}} = [u'^{(k)}\ 0]^T$;
- whether there is an agent L (e.g., RL, Inverse Dynamics Neural Network (IDNN), etc.) or a controller π (e.g., Model Predictive control (MPC), etc.) in the real setting E.

Table 4.5 summarizes the articles that use TL and characterizes them in terms of these four factors. The table also shows the tasks and algorithms.

Table 4.5: List of the surveyed articles that apply the TL approach.

Ref.	Algorithm	Transf. info. $u^{(k)}_{\text{exch}}$	Timing	Dir.	L, π in E	Tasks
[36]	TRPO^a	$a'(k)$	C-TL	2D-TL	IDNN^e	Reacher, Half cheetah, Hopper, Humanoid, Fetch robot, Trajectory control
[145]	A3C^c	NN activation functions	1-TL	1D-TL	RL	Jaco robot manipulation
[70]	CMA-ES^b	$a(k)$, $a'(k)$	C-TL	2D-TL	RL	Humanoid bipedal locomotion
[184]	TRPO^a	$o'^{(k)}, a'^{(k)}, o^{(k)}, a^{(k)}$	C-TL	2D-TL	RL	Reacher, Hopper 2D, Reacher 2D
[65]	PPO^g	π'^{*}	1-TL	1D-TL	RL	Pusher, Striker, Ergo Reacher, ErgoShield
[89]	DQL^d	NN weights	1-TL	1D-TL	MPC^f	Quadrotor collision avoiding
[188]	DQL^d	$O', Q(x'^{(k)}, a'^{(k)})$	1-TL	1D-TL	RL	Robot object manipulation

^{a}Trust Region Policy Optimization [150], ^{b}Covariance Matrix Adaptation-Evolutionary Strategy [71], ^{c}Asynchronous Actor-Critic Agents [112], ^{d}Double Q-Learning [172], ^{e}Inverse Dynamics Neural Network, ^{f} Model Predictive Control [85]

Christiano *et al.* [36] adopted TL to perform sim-to-real control input adaptation. They assume that if simulator does not exactly replicate the real robot behavior, applying the same control input in both scenarios does not necessarily lead to same observation. However, they assume that the observation $o'^{(k)}$ obtained in simulation, is the one that should be achieved also on the real environment. Therefore, given the simulated observation $o'^{(k)}$, they employ past history to discover what real control input $a^{(k)}$ can lead to $o^{(k)} \simeq o'^{(k)}$. For this purpose, a neural network is trained in order to predict the control input $a^{(k)}$ that leads to a specific $o^{(k)}$. In particular, it is assumed that a simulated-based policy π', a forward dynamic robot model F, and a sequence $\tau_i = \left(o^{(0)}, a^{(0)}, \ldots, o^{(i-1)}, a^{(i-1)}, o^{(i)}\right)$ of i real observations and $i-1$ real control inputs are known. Here, TRPO is used to learn π', but any L agent (not necessarily RL) could be used for the purpose. While training the policy π, the policy π' returns a control input $a'^{(i)}$ based on the provided history τ_i. However, rather than being applied to the real robot, it is sent to F and the resulting observation $o'^{(i)}$ is provided, together with τ_i, as inputs to L. Finally, the learned policy π provides the control input $a^{(i)}$ that results in $o^{(k)}$ similar to $o'^{(k)}$. Therefore, here the communication is continuous and in both directions $u_{\text{exch}}^{(i)} = [u'^{(i)} \ u^{(i)}]^T = [o'^{(i)} \ \tau_i]^T$. It is worth remarking that what is actually learned in this case is an inverse dynamic model (implemented as a neural network) that must be employed in conjunction with the controller learned in simulation. The authors of the cited study evaluate their approach first in a sim-to-sim scenario with several OpenAI Gym environments and then in a sim-to-real scenario based on a Fetch robot. Results highlight the effectiveness of such method and the relatively low number of samples required for convergence. However, the authors assume that, when the consequences of a control input applied in simulation differ from those applied on the real robot, real observations should match simulated ones, and that is not always true.

In [70], the Grounded Action Transformation (GAT) algorithm is proposed to learn a humanoid bipedal locomotion policy. Inspired by the Grounded Simulation Learning (GSL) idea, introduced in [59], they try to reproduce it in a RL framework, additionally improving some aspects. GLS is based on two main principles: *grounding* and *guide*. The former refers to making the simulator E' closer to the real robot

E, by properly modifying some parameters of E' on the basis of data collected from E. The latter consists in having an expert able to guide the optimization algorithm in finding the proper parameters of E' to be tuned. In practice, given an evaluation function J_{eval}, such as a penalty function (for example the opposite of the reward), a policy π is applied to the real robot in order to collect end-effector trajectories $\mathcal{D}$. By performing $\mathcal{D}$ both in simulation and in reality, and collecting the resulting real end-effector trajectories, an optimization problem is solved in order to find those E' parameters able to minimize the Kullback-Leibler divergence between the probability of observing the same trajectories in the two cases. The resulting E' is therefore used in order to find a set of candidate policies Π_C trying to minimize J_{eval}. The optimal policy is the $\pi \in \Pi_c$ such that J_{eval} is minimized once performed on the real robot E. However, the above procedure is aimed at finding the correct values of the E' parameters. Conversely, GAT introduces an action transformation function $a'^{(k)} = m(a^{(k)})$, learned in a supervised fashion, able to map each action $a^{(k)} \in A$ into an action $a'^{(k)} \in A'$. In particular, a forward robot dynamics model is trained to compute the $x^{(k+1)}$ resulting from $a^{(k)}$. The inverse robot dynamics, instead, is trained to find the simulated action $a'^{(k)}$ able to lead the simulator in $x'^{(k+1)} = x^{(k+1)}$. The resulting procedure leads to $u_{\text{exch}}^{(k)} = [u'^{(k)} \ u^{(k)}]^T = [a'^{(k)}, a^{(k)}]^T$. Both sim-to-sim and sim-to-real experiments provide good results; however, as authors point out, the drawback of using a supervised method for $m(\cdot)$ learning is that policies are no longer effective when there are changes between training and testing distributions. Moreover, neglecting the contact dynamics can lead to simulation bias.

Wulfmeier *et al.* [184] proposed Mutual Alignment Transfer Learning (MATL), a method that relies on a Generative Adversarial Network (GAN) [66]. The main idea is similar to [36]: enforcing a similarity between observations $o'^{(k)}$ and $o^{(k)}$. Indeed, although a control sequence achieving the goal in simulation may not produce the same effects in the real environment, the corresponding sequence of simulated observations, if reproduced on the real system, can lead to task accomplishment. For the purpose, here, simulator E' and real robot E

work in parallel as generators and interact with two different agents (L' and L respectively). A discriminator D, instead, is employed (and trained along with L' and L) to classify the environment from which a sequence of observations τ_κ, provided as input, has been collected. L and L' are trained not only to maximize their respective environment reward, but also to mislead the discriminator, based on the assumption that the more the discriminator is mislead, the more "aligned" the observations are. Therefore, each time a sequence of observations τ_κ is collected, D will receive it as input and will output the probability $D(\tau_\kappa)$ that it was generated by E'. A term $\log(D(\tau_\kappa))$ is respectively added and subtracted to E and E' rewards thus encouraging misleading actions (here TRPO is the employed algorithm). The proposed solution results in an experience exchange between L and L' and a consequent alignment of the collected observations. To exploit the simulator, L' is updated M times more frequently than L, thus accelerating learning. Thereby, in this case $u_{\text{exch}}^{(k)}$ is the D output and it is equal to $[-\log(D(\tau_\kappa)) \ \ \log(D(\tau_\kappa))]^T$. Wulfmeier *et al.* [184] evaluate their approach on various RL tasks: rllab [49], OpenAI Gym, and DartEnv [26]. Results show that MATL is able to work with significantly different environments of same simulator in which only parameters variation is performed. Less encouraging results are reported when employing different simulators.

A different solution was proposed in [145], where progressive nets [144] are employed for sim-to-real information transfer. Here, by exploiting the capability of those nets to learn a tasks sequence through lateral connections, simulated knowledge can be used to avoid training from scratch on real robot. A progressive net is composed of l "columns" in which the i-th column represents an independent network of κ hidden activations. Each j-th activation $\text{act}_{j,i}$ of the i-th column is a function of the same column $j-1$-th activation ($\text{act}_{j-1,i}$) and of the $j-1$-th activation of all the previous $m < i$ columns ($\text{act}_{j,1}, \ldots, \text{act}_{i-1}$). Rusu, Vecerík, *et al.* [145] propose to use this tool in order to learn a simulated controller π', via A3C algorithm, in the first column and subsequently transfer its knowledge on real robot by means of lateral connections headed towards the second column, which represents the real agent

L. Then, the second net training (L) begins. Therefore, letting s be a simulated column, $u^{(k)}_{\text{exch}} = [u'^{(i)} \; 0]^T = [(\text{act}_{1,s}, \ldots, \text{act}_{k,s}) \; 0]^T$. The authors evaluate their approach on a robot manipulation task on Jaco arm [106] in which a visual target must be reached. The performances are compared with those obtained using a fine-tuning approach showing the superiority of progressive nets.

In [65], a Neural Augmented Simulation (NAS) approach is used to reduce the RG. A Long Short Term Memory is trained on the differences between simulated and real robot, and used to adapt the simulator on the basis of real world data. The policy π resulting from a Proximal Policy Optimization algorithm is learned on the simulated environment, whose next state at each step is adjusted by using the correction term Δ provided by the LSTM. The resulting policy, therefore, associates to each estimated value of the real robot state $x^{(i)} = x'^{(i)} + \Delta$ an action $a'^{(i)} = a^{(i)}$. The transfer is in this case only from the simulator to the real robot and $u^{(k)}_{\text{exch}} = [u'^{(i)} \; 0]^T = [\pi \; 0]^T$. The NAS has been tested in a sim-to-sim and a sim-to-real transfer. For the former, authors create an artificial RG by varying some parameters of two different simulated robotics environments of Open AI Gym, one of which was considered as the real environment. For the sim-to-real transfer, two Poppy Ergo Jr robots [139] have been used in a ErgoShield task, in which an *attacker* (one of the two involved robots) is controlled to touch as often as possible the shield attached to the end-effector of a *defender* (the other robot). The defender is able to move the shield in random poses. Experimental results show good performance both in sim-to-sim and sim-to-real transfer. Moreover, since a policy-specific fine-tuning is not required, the method can be appropriate for multi-task robotic applications.

In [89], conversely, they propose to learn a RL controller in simulation (using a deep Q-Learning approach) in which the first stages represent a visual perception module, parametrized by vector θ_{VP}. Therefore, by keeping the weights fixed, they use this module to work with real-data and predict rewards for h planned control inputs by learning a DNN. The predictor is trained, by means of a real-world data-set, in order to minimize reward prediction error. In the second phase, the real application, the predictor is used by an MPC controller. Hence, at each step, the MPC controller computes a sequence of h control inputs which maximizes the expected discounted predicted reward within an horizon h. In this context, $u^{(k)}_{\text{exch}} = [u'^{(k)}\ 0]^T = [\theta_{\text{VP}}\ 0]^T$. As prescribed by the MPC approach, only the first control input is applied; hence, the process is repeated. Kang *et al.* [89] consider a nano aerial vehicle collision avoidance task to assess the proposed solution. Moreover, the authors compare it with other approaches: simulation only, simulation with fine-tuning, simulation with fine-tuning and perception fixed, real world only, supervised and unsupervised. Their solution outperforms all others tested, and shows the best result in terms of pre-collision time.

Yuan *et al.* [188] performs an action-value function adaptation in a supervised fashion. Here, a Baxter robot is asked to solve a nonprehensile rearrangement task, i.e., the problem of pushing an object into a predefined goal pose. The proposed procedure consists of three sequential steps: (1) learning, in simulation, an optimal action-value function Q_{sim} able to select the best action a' to perform when an image of the scene is provided as observation o', (2) collecting a data-set of real and simulated observation pairs (o, o'), and (3) using it, along with the pre-trained Q_{sim}, in order to create a Q_{real} useful to adapt the agent for a real world application. In the former, a deep-Q network is used to approximate Q_{sim}. In the second step, starting from real scenes o, the obstacle and the portable object positions are used to recreate the same scenes in simulation o'. In particular, randomly setting o^0, the RL agent (learned in step (1)) is applied to the real robot in order to collect a set of $O_{\text{real}} = o^0, o^1, o^2, \ldots$ from which the respective simulated counterpart set $O_{\text{sim}} = o'^0, o'^1,' o^2, \ldots$ is created. The resulting data-set, composed of $(O_{\text{real}}, O_{\text{sim}}, Q_{\text{sim}})$ is used in order to learn a Q_{real} able to minimize a loss function defined as $r + \gamma Q_{\text{sim}} - Q_{\text{real}}$. Three different strategies have been deployed: (a) train Q_{real} keeping the Q_{sim} structure but retraining the network in its entirety; (b) use Q_{sim} as baseline and adapt only the parameters of the convolutional layers; (c) add two new fully connected layers to increase the flexibility of the network and learn their parameters and those of the convolutional layers. Therefore in this case $u_{\text{exch}}^{(k)} = [u'^{(k)}\ 0]^T = [(O_{\text{sim}}, Q_{\text{sim}}), 0]^T$. Authors compare their results with the one obtained by using the same domain randomization idea of [164]. Experimental results show that their approaches surpass the one of [164] in terms of performances and, in particular, Item (c) turns out to be the best. However, collecting data both in simulation and in reality in order to build the data-set used to learn the Q_{real} may be costly and time-consuming.

4.2.4 Discussion and promising ideas

Despite several attempts found in the literature to make sim-to-real transferable controllers, many of which associate the idea of robustness with that of sim-to-real transferability (DR and ARL), the lack of uniformity of the considered tasks does not allow to determine which solution is the more appropriate in terms of the RG. Besides, a considerable fraction of the proposed approaches were not experimentally evaluated in an actual sim-to-real scenario. A controller robust to certain model disturbances or parametric variations, is not necessarily sim-to-real transferable. In fact, if these variations and disturbances do not correctly represent the simulator inaccuracies with respect to reality, it might result not sim-to-real transferable. This suggests that an interaction with the real system during training is still needed: to this respect, TL approaches appear promising. On the other hand, the TL approaches here surveyed often lacked an assessment of the robustness. Moreover, since TL requires two successive (or simultaneous) training phases, it may be exhibit low efficiency.

Some mixed approaches exist, that borrow ideas from DR, ARL, and TL. A first attempt in merging DR and TL is proposed in [119]. Here the authors propose to learn a policy in a randomized simulation and to adapt the distribution of simulation parameters on the basis of a real-world performance.

A promising research direction to tackle the RG problem could be the meta-learning strategy application in RL, in order to quickly adapt experience gained in simulation on the real system [152, 69]. In Meta-RL, given a distribution over tasks, the agent learns an adaptive policy that maximizes the expected reward for a new task from the distribution. A recent work [92] has shown the great ability of this approach to generalize in environments totally different from those used during the training.

Another promising solution to avoid a direct RL training on the real robot seems to be the Probabilistic Inference for Learning Control (PILCO) proposed by [44]. Here, a probabilistic model of the system dynamics is learned incorporating uncertainty by using only some trial on the real system and a policy is learned through it. This solution allows to avoid the RG in the first place, by training a simulator with few real interactions and using it for the trial and error procedure. However, although the potential usefulness of PILCO and Meta-RL to cope with the RG, the former underestimates state uncertainty at future time steps [45], thus possibly leading to a decrease in performance; the latter, on the other hand, is computational demanding and needs an high number of real-world evaluations [78].

4.3 Metric for assessing RL algorithms performance

We now introduce two criteria for comparing learning algorithms based on the formal framework provided in Chapter 2: effectiveness and efficiency.

The *effectiveness* of a policy learning algorithm L is a measure of how close is the learned policy to the optimal policy π^* for the closed-loop system $\pi \leftrightarrow E$. Therefore, an intuitive definition of effectiveness might be based on a set $X_0 \subseteq X$ of initial states and measuring closeness as the average difference, across those states, of the infinite horizon discounted rewards:

$$\text{Effectiveness}(L|E, X_0) = \frac{1}{|X_0|} \sum_{x \in X_0} J_\pi(x) - J_{\pi^*}(x). \tag{4.6}$$

$J_{\pi^*}(x)$ is often unknown, since π^* is often unknown. Nevertheless, Equation (4.6) can be used to compare the effectiveness of two learning algorithms L_1 and L_2, because, for a given X_0, $J_{\pi^*}(x)$ is constant. Moreover, in practice, a *finite horizon* discounted reward $J_{\pi,T}$ may be used for estimating J_π by evolving the system from $x^{(0)} = x$ for T steps instead of for infinite steps.

The *efficiency* of a learning algorithm L is a measure of how much effort is required to obtain the output, i.e., to learn the policy π. For instance, efficiency might be based on estimates of computational complexity, actual measures of wall time on some reference hardware, or the number of iterations required to converge. When the learning takes place interacting with the environment the efficiency might be measured as the number of needed observations (sample-efficiency).

Often, there is a trade-off between effectiveness and efficiency: the greater the latter, the lower the former.

4.3.0.1 The L,ϕ-gap

Assuming that E and E' are two environments, $X_0 \subseteq X$ and $X'_0 \subseteq X'$, a *state mapping function* $m : X'_0 \to X_0$ is a function that maps any initial state x' of E' to an initial state $m(x')$ of E.

Given an environment E compatible (cf. Definition 5) with an environment E', a set of initial states X'_0, a finite horizon T, and a state mapping function m, we say that a policy π for E, E' exhibits a π-gap$_{m,X'_0,T}(\pi|E, E')$ on E with respect to E':

$$\pi\text{-gap}_{m,X'_0,T}(\pi|E, E') = \frac{1}{|X'_0|} \sum_{x' \in X'_0} J'_{\pi,T}(x') - J_{\pi,T}(m(x')), \qquad (4.7)$$

where $J_{\pi,T}(m(x'))$ is the finite horizon discounted reward of π starting from $x = m(x') \in X_0$ in E and evolving for T steps. Equation (4.7) can be employed (a) to compare two policies π_1, π_2 in terms of the π-gap they exhibit on the same E with respect to the same E' or (b) to compare two environments E'_1, E'_2 in terms of the π-gap exhibited by the same policy π on the same E with respect to each of them. However, the latter should be done with caution. Indeed, for the comparison to be meaningful, it is necessary that the two finite horizon discounted reward functions $J'_{1,\pi,T}, J'_{2,\pi,T}$ take values in the same range. In practice, a possible way to meet this requirement could be to force the same absolute scale for $J'_{1,\pi,T}, J'_{2,\pi,T}$ based on empirical evaluations.

A case relevant to the present study is the one in which an *environment mapping operator* ϕ exists such that E is compatible with $E' = \phi(E)$. Intuitively, ϕ corresponds to modeling a real system described by E as a simulated system described by E' such that a policy learned on E' can be applied to E. In this scenario, given a set of initial states X'_0, a state mapping function m, a finite time horizon T, a policy learning algorithm L, and an environment mapping operator ϕ; we say that L, ϕ exhibit a L,ϕ-gap$_{m,X'_0,T}(L, \phi|E)$ on E:

$$L,\phi\text{-gap}_{m,X'_0,T}(L, \phi|E) = \pi\text{-gap}_{m,X'_0,T}(L(\phi(E))|E, \phi(E)). \qquad (4.8)$$

The L,ϕ-gap measures the difference, in terms of discounted reward, of a policy learned on an environment and applied to another environment. When the former is a simulated system and the latter the corresponding real system, L,ϕ-gap measures the *reality gap*.

The semantics of the L,ϕ-gap is as follows: it is greater than 0 if the finite horizon discounted reward of the learned policy is larger on the simulated system than on the real system and lower than 0 otherwise. In general, the lower the L,ϕ-gap, the better the L, ϕ pair. However, since the reward functions h, h' of the real and simulated systems may be different, it is not possible to define, in general, an absolute threshold of acceptability for the L,ϕ-gap. For fixed ϕ, E', and hence for a fixed pair of reward functions, the threshold ϵ can instead be subjectively set. In that case we say that L,ϕ-gap is *tolerable* if:

$$L,\phi\text{-gap}_{m,X_0',T}(L, \phi|E) \leq \epsilon \tag{4.9}$$

and that the learned policy $L(\phi(E))$ is $\phi(E)$-to-E *transferable*.

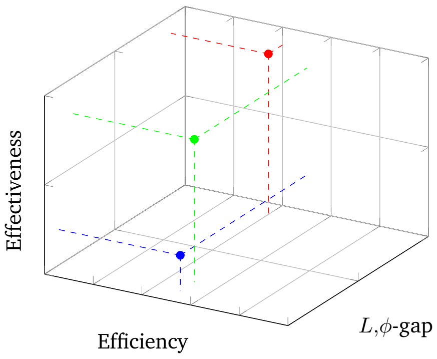


Figure 4.7: Comparison of three learning algorithms (colored dots) using three criteria: effectiveness, efficiency, and L,ϕ-gap. Different trade-offs, besides the one between effectiveness and efficiency, can be appreciated.

We propose to use L,ϕ-gap as a further criterion for assessing a learning algorithm, besides effectiveness and efficiency. Differently than the indexes related to those two criteria, the L,ϕ-gap depends, in general, also on the way the real system is modeled, i.e., on ϕ. Nevertheless, the L,ϕ-gap can be used to compare two algorithms L_1, L_2 by using the same ϕ or the same set Φ of mapping operators, in the same way a set of different environments (representing problems or tasks) is usually considered for comparing two learning algorithms. Interestingly, the L,ϕ-gap can also be used to compare two mapping operators ϕ_1, ϕ_2 (representing modeling strategies) by using the same L or a set of learning algorithms.

We think that employing the three criteria (effectiveness, efficiency, and proneness to exhibit the RG, i.e., the L,ϕ-gap) may result in more thorough assessment and comparison of learning algorithms in the context of robotics, possibly discovering new trade-offs, beside the one between effectiveness and efficiency, that might foster new research and practical achievements. Figure 4.7 graphically represents the comparison of three algorithms according to the proposed three criteria.

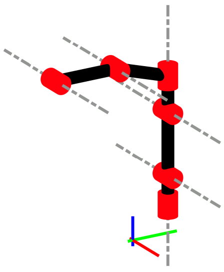
4.4 A practical example of reality gap

In the following, we deal with the sim-to-real transfer of a RL controller π for a Poppy Ergo-Jr robotic arm involved in a positioning task: i.e., moving the servo joints in order to reach a desired target position with the end-effector. In particular, we want to investigate the differences between the real robot and its simulator, and how they affect the controller performance after its transfer from the simulator to the real platform. We are aimed by the idea that a proper recognition of the differences between simulator and physical robot can assist in identifying approaches that lead to a more effective, albeit platform-dependent, solution. In the following, we focus on this latter aspect. At best, it may highlight solvable limitations of the simulator that do not lead to an increase in computation time when fixed.

For the purpose, we employ the Trust Region Policy Optimization (TRPO) [150] algorithm, which is able to perform the gradient descent optimization method, but properly manages the gradient descent step size. It belongs to the family of Actor Critic methods. For more details and the algorithm pseudocode see [150].

4.4.1 Methods

The six degrees of freedom Poppy Ergo-Jr (Figure 4.8b) is an open-source low cost educational robotic arm, designed in order to be easy to


(a) Kinematics chain

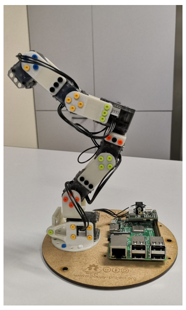

(b) The robot

Figure 4.8: Poppy Ergo-Jr and its kinematic chain.

build and modify [97]. It is composed of 6 servo motors and 3D printed elements. The resulting kinematic chain is shown in Figure 4.8a.

We denote by $q_i^{(k)} \in Q \subset \mathbb{R}$ the angular position of the i-th joint, and $\dot{q}_i^{(k)} \in \dot{Q} \subset \mathbb{R}$ the angular speed of the i-th joint at the k-th time instant, where Q and $\dot{Q}$ are respectively the set of admissible joint angular positions and speeds. Conversely we denote by $p_e^{(k)} \in \mathbb{R}^3$ the Cartesian position of the end-effector at the k-th time instant. At each k-th time instant, the input provided to π includes each joint angular position and angular velocity, along with the desired Cartesian coordinates of the end-effector $p_{e_T} \in P_{e_T} \subset \mathbb{R}^3$. The latter input allows us to get a multi-task π, i.e., a controller able to reach a predefined set of targets. The π outputs are the new suggested angular joints positions. The resulting π will therefore perform joints position control with a preset steady speed.

In order to study the sim-to-real transfer effects, we train π on the PyBullet simulator of the Poppy Ergo-Jr[3], thus testing the resulting π first on the simulator and, then, on the real platform. Finally, we compare the resulting performances. Unlike those works focused on the design of sim-to-real transferable controllers, our aim is to identify those modeling differences between the simulator and the real robot affecting the controller effectiveness. Therefore, we first of all carried out a

[3]`https://github.com/fgolemo/gym-ergojr`

systematic test, useful in understanding those model parameters whose effect is not well reproduced by the simulator. Then, we learn different π on the simulator for different values of the detected parameters. Finally we test π both on the simulator and on the real robot in order to understand how those parameters affect the π performances.

The whole procedure is described in the following.

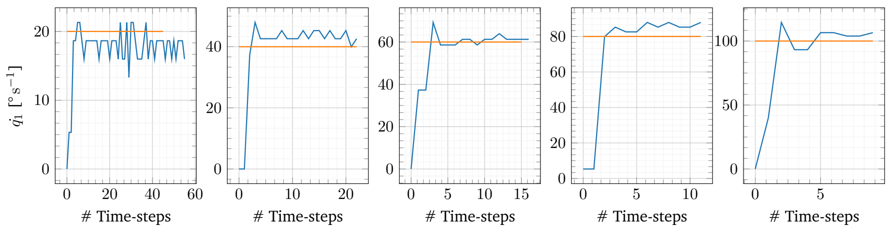


Figure 4.9: q_1 detected speed on the simulator (orange) and on the real robot (blue).

4.4.1.1 Single motor preliminary test

The servo motors composing the Poppy Ergo-Jr chain use the PID controller as a main control method[4]. This means that by imposing the speed of a joint we are simply setting the reference for its internal PID controller. The simulator should reflect this particular behavior. If it does not, or if the gains of the internal PID are set differently, it is reasonable to assume that this difference could contribute in increasing the gap in performance between the controller applied on the simulator and on the real system. Therefore, we carried out a systematic test to detect the presence of such a difference in the behavior of the simulated and real servo motors.

By keeping the T_s fixed at a predefined value, we perform a repeated movement of the first joint from an initial angular position to a final angular position, by varying, at each repetition, the motors speed among a set of different values. As evidence of the observed differences, we kept track of both simulated and real joint velocity values at each

[4] `https://emanual.robotis.com/docs/en/dxl/x/xl320/#control-mode`

time-step. Moreover, since π is a discrete-time controller, it can change the imposed motor joint speed only at each k-th time instant. Therefore, we also examined the difference between the simulated joint angular position and the real one after the first time-step of each test. The results are reported in Section 4.4.2.

4.4.1.2 TRPO training and test

Let $q_i^{(0)}$ be the initial i-th joint angular position, randomly chosen from the set of initial positions $Q_{\text{init}} \subseteq Q$.

We define the *initial condition* as the vector of joints initial angular positions $q_0 = [q_1^{(0)}, \dots, q_6^{(0)}]$.

Let p_{e_T} be the target position randomly chosen from P_{e_T}. We want to learn a π able to lead the end-effector position $p_e^{(k)}$ in the interior of a goal sphere having center in p_{e_T} and radius r_s, starting from q_0.

The performed π training procedure is episodic. Each training episode starts imposing q_0 and p_{e_T} (both randomly chosen), and terminates if at least one of the following conditions occurs: (1) the achievement of the maximum number of time-steps per episode (l_E), (2) or the task achievement. The training ends when the maximum number of cumulated time-steps is reached (l_T).

Conversely, the test procedure consists of a set of episodes performed starting from a set of n random initial conditions $\{q_0^1, \dots, q_0^n\}$. For each initial condition, the robot must reach a target position belonging to a set of m different random end-effector positions $\{p_{e_T}^1, \dots, p_{e_T}^m\}$. A successful episode is one that ends with goal achievement. We characterize the decision process of the end-effector positioning controller as $E = (X, A, O, f, g, h)$ with $f = g$. As for the state, action and reward, which completely specify the reinforcement learning setting, they are described next.

State

The state is a vector $x^{(k)} \in X$ composed of $q_i^{(k)}$ $\forall i \in \{1,2,3,4,5,6\}$ and $\dot{q}_i^{(k)}$ $\forall i \in \{1,2,3,4,5,6\}$ at the k-th time instant, and a randomly chosen p_{e_T}. An initial state $x^{(0)} \in X_0 \subseteq X$ is a state vector in which $\dot{q}_i^{(k)} = 0$ $\forall i \in \{1,2,3,4,5,6\}$ since motors start from rest.

Action

The action at the k-th time instant $a^{(k)} \in \mathbb{R}^6$ is a vector of joints positions. Note that each joint speed is fixed, hence, the position specified by the action only provides a displacement direction. Moreover, there is no guarantee that the joint positions will be reached in a single time-step.

Reward

The immediate reward $r^{(k+1)} \in \mathbb{R}$ of an action choice $a^{(k)}$ is evaluated at the $k+1$-th time instant. Let $p_e^{(k+1)}$ be the end-effector position at the $k+1$-th time instant, and p_{e_T} the center of the goal sphere; the reward is computed according to:

$$r^{(k+1)} = \begin{cases} 1 & \text{if } \left\| p_e^{(k+1)} - p_{e_T} \right\|_2 \leq r_s \\ -\left\| p_e^{(k+1)} - p_{e_T} \right\|_2 & \text{otherwise} \end{cases} . \tag{4.10}$$

The former condition denotes the goal achievement, thus the end of the episode.

4.4.1.3 Sim-to-real comparison

In order to highlight the differences between simulator and real system we perform several tests with different q_0 and p_{e_T}, and we use three different indices for comparison:

- the Effectiveness$(L|E, X_0^R)$ (Equation (4.6));
- the L,ϕ-gap$_{m,X_0^S,T}(L, \phi|E)$ (Equation (4.8));
- the error trajectory $E^{(k)}$ of the end-effector during tests.

The former implies a comparison of the average cumulated reward achieved in simulation and on the real robot, for a set of initial states X_0^R on the real robot. The second quantifies the gap of the average cumulated reward obtained in simulation and on the real robot, under the assumption that a state mapping function $m : X_0^S \to X_0^R$ exists, where X_0^S is a set of initial states in simulation. The latter, on the contrary, is independent of the reward, and is employed to compare the simulated and real trajectories achieved under the same policy. More precisely, given a simulated and a real tests, both performed starting from the same initial condition q_0 and having the same goal settings p_{e_T}, we denote by $p_{e_S}^{(k)}$ and $p_{e_R}^{(k)}$ the end-effector Cartesian coordinates at the k-th time instant, respectively on the simulator and on the real robot.

Let l_S be the duration in time-steps of a test episode on the simulator, and l_R the corresponding duration on the real robot.

We can define the end-effector trajectories in the simulated and in the real test respectively as: $P_{e_S} = \{p_{e_S}^{(0)}, p_{e_S}^{(1)}, ...p_{e_S}^{(L_S)}\}$ and $P_{e_R} = \{p_{e_R}^{(0)}, p_{e_R}^{(1)}, ...p_{e_R}^{(L_R)}\}$. The error trajectory of the end-effector during tests can be defined as:

$$E^{(k)} = p_{e_S}^{(k)} - p_{e_R}^{(k)} \quad \forall k \in [0; l_{\max}] \tag{4.11}$$

where $l_{\text{max}} = \max(l_S, l_R)$. In case of $l_S \neq l_R$, the trajectory having the lower number of samples is prolonged, in order to reach the same length of the longer one by keeping the end-effector fixed in its final position.

4.4.2 Experimental results

The following section shows in detail the results obtained performing the above depicted test procedures. In particular, reported results are oriented in highlighting the differences in performance of the controller applied on the simulator and on the real robot. The main goal is to find a relation between the discovered physical differences and those in performance.

4.4.2.1 Systematic test

We perform the test described in Section 4.4.1.1 by varying the speed of the first joint ($\dot{q}_1$) in a set of fixed values ($\dot{q}_1 \in \{20, 40, 60, 80, 100\}^\circ\,\mathrm{s}^{-1}$), while $T_s = 0.1\,\mathrm{s}$. As highlighted in Figure 4.9, the real robot has a rise and settling time not appearing in the simulator, which exhibits an ideal behavior. In particular, the simulated motor requires fewer time-steps to execute the task. However, as highlighted in Figure 4.10 the inconsistency seems to change as the set speed varies. In particular, at the speed of $60\,^\circ\,\mathrm{s}^{-1}$, the behavior between simulator and real physical system seems to be the closest.

The following tests have been conducted in order to verify whether the above mentioned inconsistency is also reflected in the performance gap of the controller learned on the simulator and tested on the real system. For this purpose, we select 3 speed values among those previously defined ($\{20, 60, 80\}^\circ\,\mathrm{s}^{-1}$), and we used them for all the performed tests. We denote this set as $\dot{Q}_{\text{test}}$.

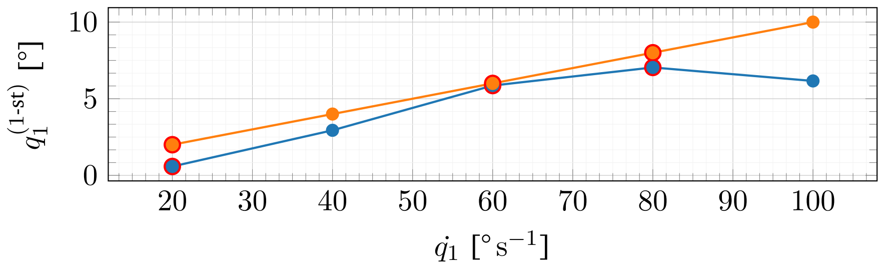


Figure 4.10: Comparison of the detected position at first time step $q_1^{(1)}$, given a reference speed, in the simulator (orange) and on the real robot (blue). Red dots highlight the speed used for performing the sim-to-real transfer.

4.4.2.2 TRPO ON SIMULATOR

We perform 10 different trainings for each $\dot{q} \in \dot{Q}_{\text{test}}$, while T_s is kept unchanged ($T_s = 0.1\,\text{s}$). At the beginning of each training episode an initial condition q_0 and a positions of the goal sphere p_{e_T} are randomly chosen in their respective set (Q_{init} and P_{e_T}). The training outputs 10 different controllers for each considered angular joint speed. In the training, we set the hyperparameters to $l_T = 2\,000\,000$, $l_E = 100$, $r_s = 0.01\,\text{m}$, and $\delta = 0.01$.

At the end of each training session we perform 100 tests for each combination of 10 different initial conditions $Q_{\text{test}} := \{q_0^1, \ldots, q_0^{10}\}$ and 10 different positions of the goal sphere $P_{e_{T\text{test}}} := \{p_{e_T}^1, \ldots, p_{e_T}^{10}\}$. Neither the initial conditions nor the targets belong to the set employed during training. Figure 4.12 summarizes Q_{test} and $P_{e_{T\text{test}}}$. In particular, each i-th subfigure shows a picture of the simulator robot in the q_0^i joints configuration and the goal sphere centered in the $p_{e_T}^i$ position.

We collect a data-set in which we store, for each training, the elapsed time (t_{learning}), and for each test the final cumulative reward ($J_\pi^S(x^{(0)})$). Figure 4.11 reports the mean and the standard deviation of the inverse of the training elapsed time (left) and the final cumulative reward obtained in test for each joint speed (right).

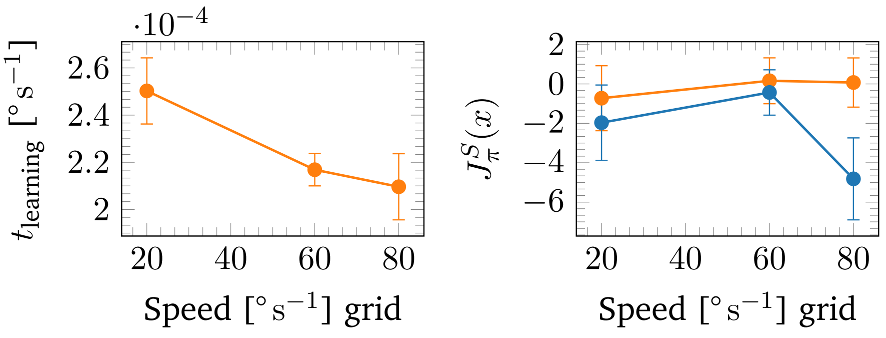


Figure 4.11: Evaluation of simulator training and test: mean and standard deviation (left) of the inverse of the training time required to TRPO for each speed; mean and standard deviation (right) of the cumulated reward obtained testing the controller on the simulator (orange) and on the real platform (blue) for each speed.

4.4.2.3 TRPO on Poppy Ergo-Jr

The total number of tests performed in simulations is $3 \times 10 \times 10 \times 10 = 3000$. Figure 4.13 summarizes the results obtained in test. In particular, each subplot is related to one of the speeds in $\dot{Q}_{\text{test}}$ (ascending order). The rows and the columns of each subplots represent respectively the indices of Q_{test} and of $P_{e_{T\text{test}}}$. The color of each square in the plot shows how many of the ten pre-trained policies achieve the goal in the simulated test.

In order to reduce the number of tests to be performed on the real platform, and therefore used for a comparison, we select only those combinations of the elements of Q_{test} and $P_{e_{T\text{test}}}$ resulting in a successful simulated test (i.e., a test terminated with task achievement) for all the policies and all the elements of $\dot{Q}_{\text{test}}$ (Figure 4.14). The resulting number of tests to be performed is $3 \times 10 \times 3 \times 10 = 900$, reducing the number of goal conditions of Figure 4.12 from 10 to 3. Precisely, the considered goal conditions are $P^R_{e_{T\text{test}}} = \{p^1_{e_T}, p^3_{e_T}, p^9_{e_T}\}$.

The mean and the standard deviation of the final cumulative reward obtained in tests for each joint speeds is reported in the blue plot of Figure 4.11. The number of successful real tests is shown in Figure 4.15.

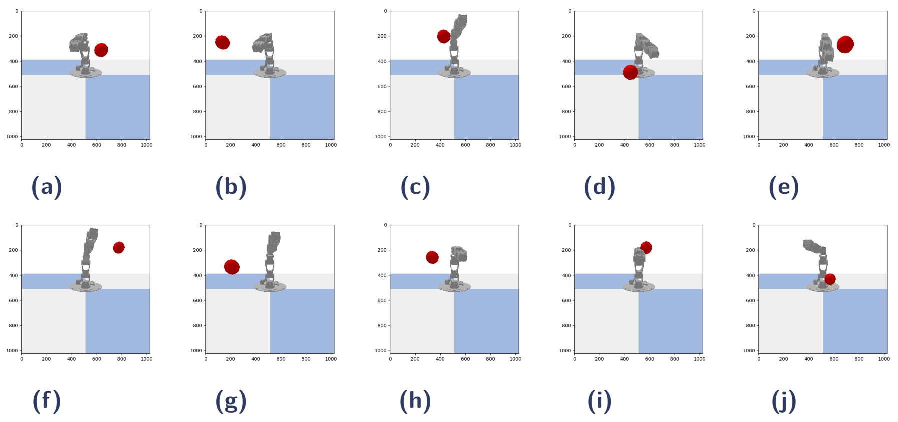


Figure 4.12: Overview of the initial joints angular position and final goal (red ball) in tests: (a)1-st angular joints initial position and 1-st goal; (b) 2-nd angular joints initial position and 2-nd goal; (c) 3-rd angular joints initial position and 3-rd goal; (d) 4-th angular joints initial position and 4-th goal; (e) 5-th angular joints initial position and 5-th goal; (f) 6-th angular joints initial position and 6-th goal; (g) 7-th angular joints initial position and 7-th goal; (h) 8-th angular joints initial position and 8-th goal; (i) 9-th angular joints initial position and 9-th goal; (j) 10-th angular joints initial position and 10-th goal.

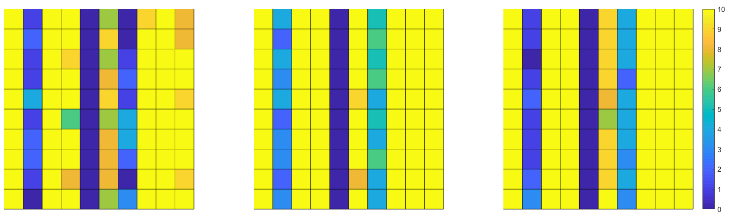


Figure 4.13: Number of success on the simulated platform for each initial (row) and goal condition (column), given the ten policies learned. Each subplot is referred to a different motor speed $(20, 60, 80)$. Yellow$=$ $10/10$ successful tests, blue$=0/10$ successful tests.

4.4.2.4 Comparisons

A first evidence of the presence of a performances gap between simulated and real robot is shown in Figure 4.15. Indeed, although the policies are tested considering only those initial conditions and target conditions that led to the task achievement in simulated tests,

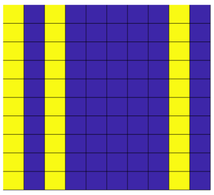


Figure 4.14: Initial-goal condition combination that results in only goal achievement in simulation tests for all the speed (yellow).

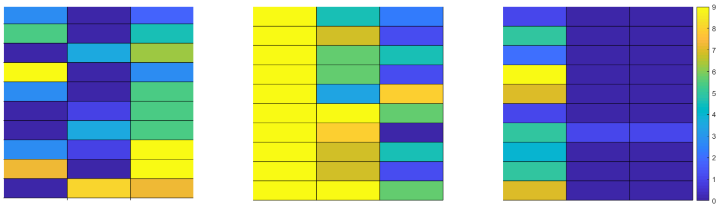


Figure 4.15: Number of success on the real platform for each initial-goal condition combination, given the ten policies. Each subplot is referred to a different motor speed $(20, 60, 80)$. Yellow= $10/10$ successful tests, blue=$0/10$ successful tests.

Figure 4.15 shows that on the real robot the result is not the same. Moreover, it suggests that the highest number of successes are achieved at the speed of $60\,^{\circ}\,\mathrm{s}^{-1}$, thus providing a first validation of the above highlighted thesis that this speed leads to closest behaviors between the simulator and the real robot.

A further evidence is highlighted in Figure 4.16, where the performance discrepancy between simulated and real tests are shown by using the effectiveness index (Equation (4.6)). In particular, it shows the mean and the standard deviation of the considered index for each speed set.

For all the considered speeds, this discrepancy is clearly evident and differs in magnitude. This is also summarized in Figure 4.17 showing the mean and the standard deviation of the $L, \phi - gap$ for each speed. In particular, when the speed is set to $60\,^{\circ}\,\mathrm{s}^{-1}$, results seem to exhibit lower performance discrepancy, and therefore a lower $L, \phi - gap$, compared to the other considered speeds. This is confirmed by observing the behavior of (Equation (4.11)).

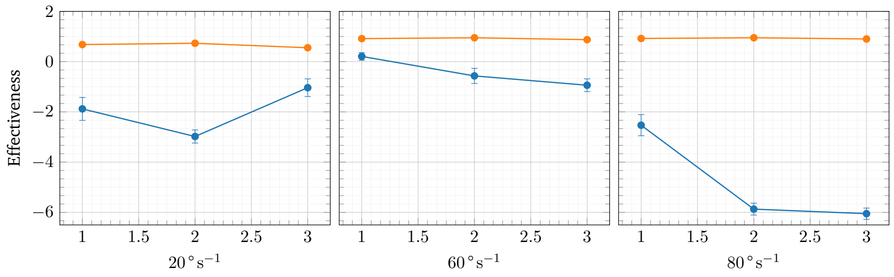


Figure 4.16: Comparison of simulated and real tests performances on the basis of the mean and the standard deviation of the Effectiveness.

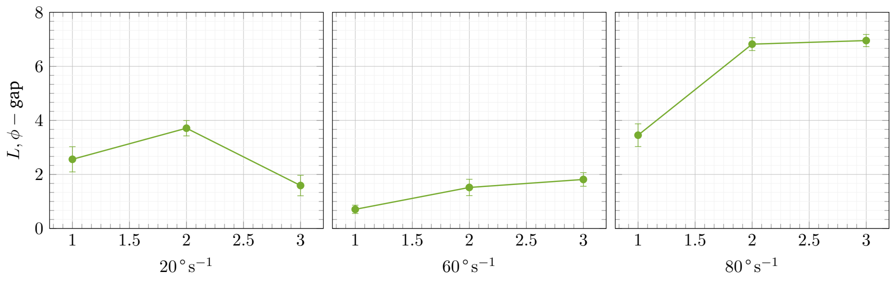


Figure 4.17: $L, \phi - \text{gap}$ evaluation of controllers learned and tested at the three different speeds value.

The box plot in Figure 4.18 shows the l_2-norm of the error trajectory:

$$\sqrt{\sum_{k=0}^{l_{max}} \left|E^{(k)}\right|^2}. \tag{4.12}$$

Here, the median value of the l_2-norm of the error for a speed of $60\,^\circ\,\mathrm{s}^{-1}$ is smaller than the one of the remaining speeds.

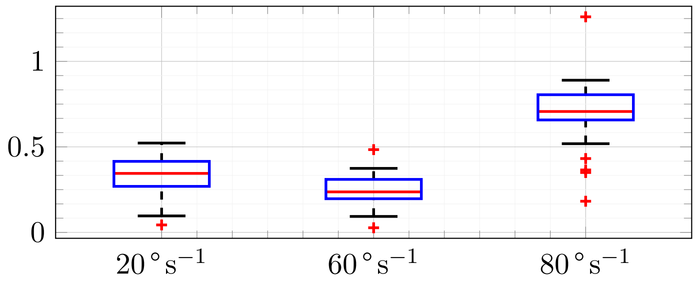


Figure 4.18: Comparison of simulated and real tests performances on the basis of a statistics of Equation (4.12) for each speed.

Therefore, as evidenced by results, the obtained controllers exhibit a reality gap and, as all the performed tests highlight, it is particularly affected by the motor speed settings. No conclusion can be drawn on the speed value leading to a mitigation of the reality gap, as the systematic test has been performed on the first joint only. However, since the simulator neglects the servomotor dynamics, this difference is probably the main source of controller performance degradation.

5 Conclusions

We sought to evaluate the effectiveness of RL as a control technique for real-world dynamical systems with the aim of answering the following research questions: (i) given the potential of RL as a control technique on simple dynamical systems, can it be used to control dynamical systems that address real-world practical problems? (ii) is there a way to quantify the reality gap? Can we use this measure to characterize a learning technique in a deeper way than just considering effectiveness and efficiency?

In order to address the former research question, we investigated the RL behavior (1) on a large-scale dynamical system such as a mixed-autonomous traffic intersection, where large numbers of vehicles interact with the controller choices in unpredictable ways, and (2) on a physical-world grounded control system such as the FEL facility of Elettra Sincrotrone Trieste. We observed that both in the first control problem, which was addressed in a simulated environment only, and in the second case study, which, in contrast, was performed directly on the real structure, the RL agent converges to an effective control law within a reasonable time.

We then moved to the second research question, focusing our attention on the applications of RL in robotics, specifically analyzing the reality gap problem. We provided a formal framework for the RG and reviewed the most significant existing methods aiming to achieve sim-to-real transferable controllers in robotics RL applications. We categorized them in the main three approaches: domain randomization, adversarial reinforcement learning, and transfer learning. Moreover, we described them according to the proposed formal framework and in terms of the employed algorithms, involved tasks, and test typology (sim-to-real transfer or just sim-to-sim). Finally, we reported a practical example of the RG in robotics, in which we characterized some simulator modeling-errors of the Poppy Ergo-Jr platform focusing on those differences, between simulator and real robot, affecting the performances of a DRL controller trained only in simulation and tested on both simulator and real robot. As evidenced by the obtained results, the unrealistic modeling of servo motor dynamics is a major source of controller performance deterioration. However, it seems to be mitigated by a proper choice of motor reference speed.

We conclude by pointing out the significant findings and open challenges derived from this work.

As observed in Chapter 3 the proper design of RL agents can lead to effective control laws, also for real world control problems. Sometimes,(if no safety issues are expected), directly on the plant, and without the use of computationally prohibitive algorithms.

However, in those cases in which the safety can be at risk (Chapter 4), real-world control problems need an appropriate simulator of the environment to train the RL controller. A general, task-independent, approach able to guarantee an effective sim-to-real transferability of the controller is still missing. With this in mind, we believe that our $L - \phi -$ gap can significantly contribute towards the solution of this open challenge. Indeed, being able to characterize and quantify the RG would (i) enable a systematic comparison among different techniques, hence favoring the advancement of research, and (ii) allow to use the measure of RG directly as an optimization objective, hence putting the transferability as a direct goal in the learning of RG-aware controllers.

Another significant and more general open problem is the sample efficiency. As highlighted in [168], RL is very data-intensive. The computation effort required for training RL agents involved in a complex task can be huge, thus limiting the practical applicability of such methods. This issue is unquestionably aggravated in those cases in which an interaction with different environment domains is involved. To this end, the meta-learning approaches seem to be headed in the right direction [152]. Future work may include merging the concept of meta-learning and our $L-\phi-$gap index to develop approaches that are both effective and efficient.

References

[1] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. "An application of reinforcement learning to aerobatic helicopter flight". In: *Advances in neural information processing systems*. 2007, pp. 1–8.

[2] Monireh Abdoos, Nasser Mozayani, and Ana L. C. Bazzan. "Traffic light control in non-stationary environments based on multi agent Q-learning". In: *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. 2011, pp. 1580–1585.

[3] Milton Abramowitz and Irene A Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. 10th ed. Vol. 55. US Government printing office, 1972.

[4] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. "Constrained policy optimization". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 22–31.

[5] Jürgen Ackermann. *Robust control: the parameter space approach*. Springer Science & Business Media, 2012.

[6] Ilya Agapov, Gianluca Geloni, S. Tomin, and I. Zagorodnov. "OCELOT: A software framework for synchrotron light source and FEL studies". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 768 (2014), pp. 151–156.

[7] Ilya Agapov, Gianluca Geloni, and Igor Zagorodnov. "Statistical optimization of FEL performance". In: *IPAC* (2015).

[8] E Allaria, Roberto Appio, L Badano, WA Barletta, S Bassanese, SG Biedron, A Borga, E Busetto, D Castronovo, P Cinquegrana, *et al.* “Highly coherent and stable pulses from the FERMI seeded free-electron laser in the extreme ultraviolet”. In: *Nature Photonics* 6.10 (2012), p. 699.

[9] E Allaria, L Badano, S Bassanese, Flavio Capotondi, D Castronovo, Paolo Cinquegrana, MB Danailov, G D’Auria, A Demidovich, R De Monte, *et al.* “The FERMI free-electron lasers”. In: *Journal of synchrotron radiation* 22.3 (2015), pp. 485–491.

[10] E Allaria, D Castronovo, P Cinquegrana, P Craievich, Massimo Dal Forno, MB Danailov, G D’Auria, A Demidovich, G De Ninno, S Di Mitri, *et al.* “Two-stage seeded soft-X-ray free-electron laser”. In: *Nature Photonics* 7.11 (2013), p. 913.

[11] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. “Hindsight experience replay”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5048–5058.

[12] Koppaka Ganesh Sai Apuroop, Anh Vu Le, Mohan Rajesh Elara, and Bing J. Sheu. “Reinforcement Learning-Based Complete Area Coverage Path Planning for a Modified hTrihex Robot”. In: *Sensors* 21.4 (2021).

[13] Sahar Araghi, Abbas Khosravi, and Douglas Creighton. “A review on computational intelligence methods for controlling traffic signal timing”. In: *Expert systems with applications* 42.3 (2015), pp. 1538–1550.

[14] Kartik B Ariyur and Miroslav Krstić. *Real-time optimization by extremum-seeking control*. John Wiley & Sons, 2003.

[15] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. “Deep reinforcement learning: A brief survey”. In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

[16] Christopher G Atkeson and Stefan Schaal. “Robot learning from demonstration”. In: *ICML*. Vol. 97. Citeseer. 1997, pp. 12–20.

[17] Gabriella Azzopardi, Belen Salvachua, Gianluca Valentino, Stefano Redaelli, and Adrian Muscat. “Operational results on the fully automatic LHC collimator alignment”. In: *Physical Review Accelerators and Beams* 22.9 (2019), p. 093001.

[18] Abhijit S Badwe, Ravindra D Gudi, Rohit S Patwardhan, Sirish L Shah, and Sachin C Patwardhan. “Detection of model-plant mismatch in MPC applications”. In: *Journal of Process Control* 19.8 (2009), pp. 1305–1313.

[19] Richard Bellman. “On the theory of dynamic programming”. In: *Proceedings of the National Academy of Sciences of the United States of America* 38.8 (1952), p. 716.

[20] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. “Curriculum learning”. In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 41–48.

[21] Dimitri P Bertsekas. *Dynamic programming and optimal control*. Vol. 1. 2. Athena scientific Belmont, MA, 1995.

[22] Dimitri P Bertsekas. “Reinforcement learning and optimal control”. In: *Athena Scientific* (2019).

[23] SP Bhattacharyya. “Robust control under parametric uncertainty: an overview and recent results”. In: *Annual Reviews in Control* 44 (2017), pp. 45–77.

[24] Blender Community. *Blender: Open Source 3D modeling suit.* `http://www.blender.org.`. Online. 2020.

[25] Michel Breyer, Fadri Furrer, Tonci Novkovic, Roland Siegwart, and Juan Nieto. “Flexible robotic grasping with sim-to-real transfer based reinforcement learning”. In: *ArXiv e-prints* (2018).

[26] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. “Openai gym”. In: *arXiv preprint: 1606.01540* (2016).

[27] Niky Bruchon, Gianfranco Fenu, Giulio Gaio, Marco Lonza, Finn Henry O’Shea, Felice Andrea Pellegrino, and Erica Salvato. “Basic reinforcement learning techniques to control the intensity of a seeded free-electron laser”. In: *Electronics* 9.5 (2020), p. 781.

[28] Niky Bruchon, Gianfranco Fenu, Giulio Gaio, Marco Lonza, Felice Andrea Pellegrino, and Erica Salvato. “Toward the Application of Reinforcement Learning to the Intensity Control of a Seeded Free-Electron Laser”. In: *2019 23rd International Conference on Mechatronics Technology (ICMT)*. Ed. by Adolfo Senatore and Truong Q. Dinh. Salerno: IEEE, Oct. 2019, pp. 1–6.

[29] Niky Bruchon, Gianfranco Fenu, Giulio Gaio, Marco Lonza, Felice Andrea Pellegrino, and Lorenzo Saule. “Free-electron laser spectrum evaluation and automatic optimization”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 871 (2017), pp. 20–29.

[30] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. “Sample-efficient reinforcement learning with stochastic ensemble value expansion”. In: *arXiv preprint: 1807.01675* (2018).

[31] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2017.

[32] Dan A Calian, Daniel J Mankowitz, Tom Zahavy, Zhongwen Xu, Junhyuk Oh, Nir Levine, and Timothy Mann. “Balancing Constraints and Rewards with Meta-Gradient D4PG”. In: *arXiv preprint: 2010.06324* (2020).

[33] K. Chatzilygeroudis, V. Vassiliades, F. Stulp, S. Calinon, and J. Mouret. “A Survey on Policy Search Algorithms for Learning Robot Controllers in a Handful of Trials”. In: *IEEE Transactions on Robotics* (2019), pp. 1–20.

[34] Suh-Wen Chiou. “A robust signal control system for equilibrium flow under uncertain travel demand and traffic delay”. In: *Automatica* 96 (2018), pp. 240–252.

[35] Samuel PM Choi and Dit-Yan Yeung. “Predictive Q-routing: A memory-based reinforcement learning approach to adaptive

traffic control”. In: *Advances in Neural Information Processing Systems*. 1996, pp. 945–951.

[36] Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba. “Transfer from simulation to real world through learning deep inverse dynamics model”. In: *arXiv preprint: 1610.03518* (2016).

[37] T. Chu and J. Wang. “Traffic signal control by distributed Reinforcement Learning with min-sum communication”. In: *2017 American Control Conference (ACC)*. 2017, pp. 5095–5100.

[38] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. “Deep reinforcement learning in a handful of trials using probabilistic dynamics models”. In: *arXiv preprint: 1805.12114* (2018).

[39] S Cleva, L Pivetta, P Sigalotti, *et al.* “BeagleBone for embedded control system applications”. In: *Proc. ICALEPCS2013*. 2013.

[40] CM Labs. *Vortex Studio Real-time simulation and visualization software for system-level modeling of mechatronics and mechanical equipment.* `https://www.cm-labs.com/vortex-studio/`. Online. 2001.

[41] Erwin Coumans and Yunfei Bai. *Bullet Physics SDK*. `http://bulletphysics.org`. 2021.

[42] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. `http://pybullet.org`. 2016.

[43] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. “Safe exploration in continuous action spaces”. In: *arXiv preprint: 1801.08757* (2018).

[44] Marc Deisenroth and Carl E Rasmussen. “PILCO: A model-based and data-efficient approach to policy search”. In: *Proceedings of the 28th International Conference on machine learning (ICML-11)*. 2011, pp. 465–472.

[45] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. “Gaussian processes for data-efficient learning in robotics and control”. In: *IEEE transactions on pattern analysis and machine intelligence* 37.2 (2013), pp. 408–423.

[46] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, *et al.* “A survey on policy search for robotics”. In: *Foundations and Trends® in Robotics* 2.1–2 (2013), pp. 1–142.

[47] J. C. Doyle, K. Glover, P. P. Khargonekar, and B. A. Francis. “State-space solutions to standard H/sub 2/ and H/sub infinity / control problems”. In: *IEEE Transactions on Automatic Control* 34.8 (Aug. 1989), pp. 831–847.

[48] Kurt Dresner and Peter Stone. “Multiagent traffic management: A reservation-based intersection control mechanism”. In: *Autonomous Agents and Multiagent Systems, International Joint Conference on*. Vol. 3. IEEE Computer Society. 2004, pp. 530–537.

[49] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. “Benchmarking deep reinforcement learning for continuous control”. In: *International Conference on Machine Learning*. 2016, pp. 1329–1338.

[50] Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. “Challenges of real-world reinforcement learning: definitions, benchmarks and analysis”. In: *Machine Learning* (2021), pp. 1–50.

[51] H. Durrant-Whyte, N. Roy, and P. Abbeel. “Learning to Control a Low-Cost Manipulator Using Data-Efficient Reinforcement Learning”. In: *Robotics: Science and Systems VII*. MITP, 2012, pp. 57–64.

[52] A. L. Edelen, S. G. Biedron, B. E. Chase, D. Edstrom, S. V. Milton, and P. Stabile. “Neural networks for modeling and control of particle accelerators”. In: *IEEE Transactions on Nuclear Science* 63.2 (2016), pp. 878–897.

[53] Auralee Edelen, Christopher Mayes, Daniel Bowring, Daniel Ratner, Andreas Adelmann, Rasmus Ischebeck, Jochem Snuverink, Ilya Agapov, Raimund Kammering, Jonathan Edelen, *et al.* “Opportunities in machine learning for particle accelerators”. In: *arXiv preprint: 1811.03172* (2018).

[54] Auralee Edelen, Nicole Neveu, Matthias Frey, Yannick Huber, Christopher Mayes, and Andreas Adelmann. “Machine learning for orders of magnitude speedup in multiobjective optimization of particle accelerator systems”. In: *Physical Review Accelerators and Beams* 23.4 (2020), p. 044601.

[55] Auralee L Edelen, Jonathan P Edelen, LLC RadiaSoft, Sandra G Biedron, Stephen V Milton, and Peter JM van der Slot. “Using Neural Network Control Policies For Rapid Switching Between Beam Parameters in a Free-Electron Laser”. In: *NIPS* (2017).

[56] Auralee L. Edelen, Stephen Val Milton, Sandra G. Biedron, Jonathan P. Edelen, and Peter J. M. van der Slot. *Using A Neural Network Control Policy For Rapid Switching Between Beam Parameters in an FEL*. Tech. rep. Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2017.

[57] C Emma, A Edelen, MJ Hogan, B O’Shea, G White, and V Yakimenko. “Machine learning-based longitudinal phase space prediction of particle accelerators”. In: *Physical Review Accelerators and Beams* 21.11 (2018), p. 112802.

[58] Myungeun Eom and Byung-In Kim. “The traffic signal control problem for intersections: a review”. In: *European transport research review* 12.1 (2020), pp. 1–20.

[59] Alon Farchy, Samuel Barrett, Patrick MacAlpine, and Peter Stone. “Humanoid robots learning to walk faster: From the real world to simulation and back”. In: *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. 2013, pp. 39–46.

[60] Julia L. Fleck, Christos G. Cassandras, and Yanfeng Geng. “Adaptive Quasi-Dynamic Traffic Light Control”. In: *IEEE Transactions on Control Systems Technology* 24.3 (2016), pp. 830–842.

[61] Dario Floreano and Joseba Urzelai. “Evolution of plastic control networks”. In: *Autonomous robots* 11.3 (2001), pp. 311–317.

[62] Elena Fol, JM Coello de Portugal, Giuliano Franchetti, and Rogelio Tomás. “Optics corrections using Machine Learning in the LHC”. In: *Proceedings of the 2019 International Particle Accelerator Conference, Melbourne, Australia*. 2019.

[63] Giulio Gaio, Marco Lonza, Niky Bruchon, and Lorenzo Saule. “Advances in Automatic Performance Optimization at FERMI”. In: *ICALEPCS* (2018).

[64] Alborz Geramifard, Thomas J Walsh, Stefanie Tellex, Girish Chowdhary, Nicholas Roy, Jonathan P How, *et al.* “A tutorial on linear function approximators for dynamic programming and reinforcement learning”. In: *Foundations and Trends® in Machine Learning* 6.4 (2013), pp. 375–451.

[65] Florian Golemo, Adrien Ali Taiga, Aaron Courville, and Pierre-Yves Oudeyer. “Sim-to-real transfer with neural-augmented robot simulation”. In: *Conference on Robot Learning*. 2018, pp. 817–828.

[66] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. “Generative adversarial nets”. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[67] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.

[68] Vijaykumar Gullapalli, Judy A Franklin, and Hamid Benbrahim. “Acquiring robot skills via reinforcement learning”. In: *IEEE Control Systems Magazine* 14.1 (1994), pp. 13–24.

[69] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. “Meta-reinforcement learning of structured exploration strategies”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 5302–5311.

[70] Josiah P Hanna and Peter Stone. "Grounded action transformation for robot learning in simulation". In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[71] Nikolaus Hansen. "The CMA evolution strategy: a comparing review". In: *Towards a new evolutionary computation*. Springer, 2006, pp. 75–102.

[72] Cédric Hartland and Nicolas Bredeche. "Evolutionary robotics, anticipation and the reality gap". In: *2006 IEEE International Conference on Robotics and Biomimetics*. IEEE. 2006, pp. 1640–1645.

[73] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. "Memory-based control with recurrent neural networks". In: *arXiv preprint: 1512.04455* (2015).

[74] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, *et al.* "Deep q-learning from demonstrations". In: *Thirty-second AAAI conference on artificial intelligence*. 2018.

[75] Simon Hirlaender, Verena Kain, and Michael Schenk. *New paradigms for tuning accelerators: Automatic performance optimization and first steps towards reinforcement learning at the CERN Low Energy Ion Ring*. URL: `https://indico.cern.ch/event/784769/contributions/3265006/attachments/1807476/2950489/CO-technical-meeting-_Hirlaender.pdf`. 2nd ICFA Workshop on Machine Learning for Charged Particle Accelerators. 2019.

[76] Benjamin F Hobbs and Ann Hepenstal. "Is optimization optimistically biased?" In: *Water Resources Research* 25.2 (1989), pp. 152–160.

[77] Sebastian Höfer, Kostas Bekris, Ankur Handa, Juan Camilo Gamboa, Melissa Mozifian, Florian Golemo, Chris Atkeson, Dieter Fox, Ken Goldberg, John Leonard, *et al.* "Sim2Real in Robotics and Automation: Applications and Challenges". In: *IEEE Transactions on Automation Science and Engineering* 18.2 (2021), pp. 398–400.

[78] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. “Meta-learning in neural networks: A survey”. In: *arXiv preprint: 2004.05439* (2020).

[79] Han Hu, Kaicheng Zhang, Aaron Hao Tan, Michael Ruan, Christopher Agia, and Goldie Nejat. “A Sim-to-Real Pipeline for Deep Reinforcement Learning for Autonomous Robot Navigation in Cluttered Rough Terrain”. In: *IEEE Robotics and Automation Letters* 6.4 (2021), pp. 6569–6576.

[80] Shan Huang, Adel W Sadek, and Yunjie Zhao. “Assessing the mobility and environmental benefits of reservation-based intelligent intersections using an integrated simulator”. In: *IEEE Transactions on Intelligent Transportation Systems* 13.3 (2012), pp. 1201–1214.

[81] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. “Image-to-image translation with conditional adversarial networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.

[82] Nick Jakobi, Phil Husbands, and Inman Harvey. “Noise and the reality gap: The use of simulation in evolutionary robotics”. In: *European Conference on Artificial Life*. Springer. 1995, pp. 704–720.

[83] Stephen James, Andrew J. Davison, and Edward Johns. “Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task”. In: *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*. 2017, pp. 334–343.

[84] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. “Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12627–12637.

[85] Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. “Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–8.

[86] Verena Kain, Simon Hirlander, Brennan Goddard, Francesco Maria Velotti, Giovanni Zevi Della Porta, Niky Bruchon, and Gianluca Valentino. “Sample-efficient reinforcement learning for CERN accelerator control”. In: *Physical Review Accelerators and Beams* (2020).

[87] Sham M Kakade. “A natural policy gradient”. In: *Advances in neural information processing systems*. 2002, pp. 1531–1538.

[88] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. “Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation”. In: *Proceedings of The 2nd Conference on Robot Learning*. Ed. by Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto. Vol. 87. Proceedings of Machine Learning Research. PMLR, 2018, pp. 651–673.

[89] K. Kang, S. Belkhale, G. Kahn, P. Abbeel, and S. Levine. “Generalization through Simulation: Integrating Simulated and Real Data into Deep Reinforcement Learning for Vision-Based Autonomous Flight”. In: *2019 International Conference on Robotics and Automation (ICRA)*. May 2019, pp. 6008–6014.

[90] Manabu Kano, Yohei Shigi, Shinji Hasebe, and Satoshi Ooyama. “Detection of significant model-plant mismatch from routine operation data of model predictive control system”. In: *IFAC Proceedings Volumes* 43.5 (2010), pp. 685–690.

[91] K. V. Katsikopoulos and S. E. Engelbrecht. “Markov decision processes with delays and asynchronous cost collection”. In: *IEEE Transactions on Automatic Control* 48.4 (2003), pp. 568–574.

[92] Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. "Improving Generalization in Meta Reinforcement Learning using Learned Objectives". In: *arXiv preprint: 1910.04098* (2019).

[93] Jens Kober, J Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[94] Nate Kohl and Peter Stone. "Policy gradient reinforcement learning for fast quadrupedal locomotion". In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*. Vol. 3. IEEE. 2004, pp. 2619–2624.

[95] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. "The transferability approach: Crossing the reality gap in evolutionary robotics". In: *IEEE Transactions on Evolutionary Computation* 17.1 (2012), pp. 122–145.

[96] Sam Kriegman, Douglas Blackiston, Michael Levin, and Josh Bongard. "A scalable pipeline for designing reconfigurable organisms". In: *Proceedings of the National Academy of Sciences* 117.4 (2020), pp. 1853–1859. eprint: `https://www.pnas.org/content/117/4/1853.full.pdf`.

[97] Matthieu Lapeyre. "Poppy: open-source, 3D printed and fully-modular robotic platform for science, art and education". PhD thesis. 2014.

[98] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. "Reinforcement learning with augmented data". In: *arXiv preprint: 2004.14990* (2020).

[99] Jeongseok Lee, Michael X Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S Srinivasa, Mike Stilman, and C Karen Liu. "DART: Dynamic Animation and Robotics Toolkit." In: *J. Open Source Software* 3.22 (2018), p. 500.

[100] Z. Li, X. Cheng, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath. "Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots". In: *arXiv preprint: 2103.14295* (2021).

[101] Michael L Littman. “Markov games as a framework for multi-agent reinforcement learning”. In: *Machine learning proceedings 1994*. Elsevier, 1994, pp. 157–163.

[102] Di Liu, Wenwu Yu, Simone Baldi, Jinde Cao, and Wei Huang. “A Switching-Based Adaptive Dynamic Programming Method to Optimal Traffic Signaling”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 50.11 (2020), pp. 4160–4170.

[103] Rongrong Liu, Florent Nageotte, Philippe Zanne, Michel de Mathelin, and Birgitta Dresp-Langley. “Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review”. In: *Robotics* 10.1 (2021), p. 22.

[104] D. L. Ma and R. D. Braatz. “Worst-case analysis of finite-time control policies”. In: *IEEE Transactions on Control Systems Technology* 9.5 (Sept. 2001), pp. 766–774.

[105] D. L. Ma, S. H. Chung, and R. D. Braatz. “Worst-case performance analysis of optimal batch control trajectories”. In: *1999 European Control Conference (ECC)*. Aug. 1999, pp. 3256–3261.

[106] V. Maheu, P. S. Archambault, J. Frappier, and F. Routhier. “Evaluation of the JACO robotic arm: Clinico-economic study for powered wheelchair users with upper-extremity disabilities”. In: *2011 IEEE International Conference on Rehabilitation Robotics*. 2011, pp. 1–5.

[107] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Li Fei-Fei, and Silvio Savarese. “Adversarially robust policy learning: Active construction of physically-plausible perturbations”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3932–3939.

[108] Jan Matas, Stephen James, and Andrew J. Davison. “Sim-to-Real Reinforcement Learning for Deformable Object Manipulation”. In: *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*. 2018, pp. 734–743.

[109] Mitchell McIntire, Tyler Cope, Daniel Ratner, and Stefano Ermon. “Bayesian optimization of FEL performance at LCLS”. In: *IPAC* (2016).

[110] Mitchell McIntire, Daniel Ratner, and Stefano Ermon. “Sparse Gaussian Processes for Bayesian Optimization.” In: *UAI*. 2016.

[111] Hiroyuki Miyamoto, Stefan Schaal, Francesca Gandolfo, Hiroaki Gomi, Yasuharu Koike, Rieko Osu, Eri Nakano, Yasuhiro Wada, and Mitsuo Kawato. “A kendama learning robot based on bidirectional theory”. In: *Neural networks* 9.8 (1996), pp. 1281–1302.

[112] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. “Asynchronous methods for deep reinforcement learning”. In: *International conference on machine learning*. 2016, pp. 1928–1937.

[113] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, *et al.* “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[114] Andrew W Moore and Christopher G Atkeson. “Prioritized sweeping: Reinforcement learning with less data and less time”. In: *Machine learning* 13.1 (1993), pp. 103–130.

[115] I. Mordatch, K. Lowrey, and E. Todorov. “Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 5307–5314.

[116] Jean-Baptiste Mouret and Konstantinos Chatzilygeroudis. “20 years of reality gap: a few thoughts about simulators in evolutionary robotics”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM. 2017, pp. 1121–1124.

[117] Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. "Deep reinforcement learning: an overview". In: *Proceedings of SAI Intelligent Systems Conference*. Springer. 2016, pp. 426–440.

[118] Roland Müller, A Balzer, P Baumgärtel, OP Sauer, G Hartmann, and J Viefhaus. "Modernization of experimental data taking at BESSY II". In: *MOCPL02, Proceedings of this conference*. 2019.

[119] Fabio Muratore, Christian Eilers, Michael Gienger, and Jan Peters. "Bayesian domain randomization for sim-to-real transfer". In: *arXiv preprint: 2003.02471* (2020).

[120] Fabio Muratore, Felix Treede, Michael Gienger, and Jan Peters. "Domain Randomization for Simulation-Based Policy Optimization with Transferability Assessment". In: *Conference on Robot Learning*. 2018, pp. 700–713.

[121] Grzegorz Mzyk. "Wiener System". In: *Combined Parametric-Nonparametric Identification of Block-Oriented Systems*. Cham: Springer International Publishing, 2014, pp. 87–102.

[122] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. "Autonomous inverted helicopter flight via reinforcement learning". In: *Experimental robotics IX*. Springer, 2006, pp. 363–372.

[123] Andrew Y Ng, Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping". In: *ICML*. Vol. 99. 1999, pp. 278–287.

[124] Gustav Nilsson and Giacomo Como. "A Micro-Simulation Study of the Generalized Proportional Allocation Traffic Signal Control". In: *IEEE Transactions on Intelligent Transportation Systems* 21.4 (2020), pp. 1705–1715.

[125] Stefano Nolfi, Dario Floreano, Orazio Miglino, and Francesco Mondada. "How to evolve autonomous robots: Different approaches in evolutionary robotics". In: *Artificial life iv: Proceedings of the fourth international workshop on the synthesis and simulation of living systems*. MIT Press. 1994, pp. 190–197.

[126] FH O'Shea, N Bruchon, and G Gaio. "Policy gradient methods for free-electron laser and terahertz source optimization and stabilization at the FERMI free-electron laser at Elettra". In: *Physical Review Accelerators and Beams* 23.12 (2020), p. 122802.

[127] Laurentz E Olivier and Ian K Craig. "Model-plant mismatch detection and model update for a run-of-mine ore milling circuit under model predictive control". In: *Journal of Process Control* 23.2 (2013), pp. 100–107.

[128] Open Source Robotics Foundation. *GAZEBO: Robot simulation made easy*. `http://www.gazebosim.org/`. Online. 2020.

[129] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. "Solving Rubik's Cube with a Robot Hand". In: *arXiv preprint* (2019).

[130] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. "Deep exploration via bootstrapped DQN". In: *Advances in neural information processing systems*. 2016, pp. 4026–4034.

[131] X. Pan, D. Seita, Y. Gao, and J. Canny. "Risk Averse Robust Adversarial Reinforcement Learning". In: *2019 International Conference on Robotics and Automation (ICRA)*. May 2019, pp. 8522–8528.

[132] Markos Papageorgiou, Christina Diakaki, Vaya Dinopoulou, Apostolos Kotsialos, and Yibing Wang. "Review of road traffic control strategies". In: *Proceedings of the IEEE* 91.12 (2003), pp. 2043–2067.

[133] Jing Peng and Ronald J Williams. "Efficient learning and planning within the Dyna framework". In: *Adaptive Behavior* 1.4 (1993), pp. 437–454.

[134] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. "Sim-to-real transfer of robotic control with dy-

namics randomization”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–8.

[135] Jan Peters and Stefan Schaal. “Learning to control in operational space”. In: *The International Journal of Robotics Research* 27.2 (2008), pp. 197–212.

[136] Jan Peters and Stefan Schaal. “Natural actor-critic”. In: *Neurocomputing* 71.7-9 (2008), pp. 1180–1190.

[137] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. “Robust adversarial reinforcement learning”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 2817–2826.

[138] Jordan B Pollack, Hod Lipson, Sevan Ficici, Pablo Funes, Greg Hornby, and Richard A Watson. “Evolutionary techniques in physical robotics”. In: *International Conference on Evolvable Systems*. Springer. 2000, pp. 175–186.

[139] Poppy Station. *Poppy Project: Open Source Robotics Platform.* `https://www.poppy-project.org/en/`. Online. 2020.

[140] Alexander Radovic, Mike Williams, David Rousseau, Michael Kagan, Daniele Bonacorsi, Alexander Himmel, Adam Aurisano, Kazuhiro Terao, and Taritree Wongjirad. “Machine learning at the energy and intensity frontiers of particle physics”. In: *Nature* 560.7716 (2018), pp. 41–48.

[141] L Vera Ramirez, T Mertens, R Mueller, J Viefhaus, and G Hartmann. “Adding Machine Learning to the Analysis and Optimization Toolsets at the Light Source BESSY II”. In: *ICALEPCS* (2019).

[142] Benjamin Recht. “A tour of reinforcement learning: The view from continuous control”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 2 (2019), pp. 253–279.

[143] Andres C Rodriguez, Ronald Parr, and Daphne Koller. “Reinforcement learning using approximate belief states”. In: *Advances in Neural Information Processing Systems*. 2000, pp. 1036–1042.

[144] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. “Progressive neural networks”. In: *arXiv preprint: 1606.04671* (2016).

[145] Andrei A. Rusu, Matej Vecerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. “Sim-to-Real Robot Learning from Pixels with Progressive Nets”. In: *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*. 2017, pp. 262–270.

[146] Fereshteh Sadeghi and Sergey Levine. “Cad2rl: Real single-image flight without a single real image”. In: *arXiv preprint: 1611.04201* (2016).

[147] Erica Salvato, Gianfranco Fenu, Eric Medvet, and Felice Andrea Pellegrino. “Characterization of Modeling Errors Affecting Performances of a Robotics Deep Reinforcement Learning Controller in a Sim-to-Real Transfer”. In: *2021 44rd International Convention on Information, Communication and Electronic Technology (MIPRO)*. 2021.

[148] Erica Salvato, Gianfranco Fenu, Eric Medvet, and Felice Andrea Pellegrino. “Crossing the Reality Gap: a Survey on Sim-to-Real Transferability of Robot Controllers in Reinforcement Learning”. In: (2021).

[149] Harsh Satija, Philip Amortila, and Joelle Pineau. “Constrained markov decision processes via backward value functions”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 8502–8511.

[150] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. “Trust region policy optimization”. In: *International conference on machine learning*. 2015, pp. 1889–1897.

[151] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. “Proximal policy optimization algorithms”. In: *arXiv preprint: 1707.06347* (2017).

[152] Nicolas Schweighofer and Kenji Doya. “Meta-learning in reinforcement learning”. In: *Neural Networks* 16.1 (2003), pp. 5–9.

[153] S Selvanathan and AK Tangirala. “Diagnosis of poor control loop performance due to model- plant mismatch”. In: *Industrial & Engineering Chemistry Research* 49.9 (2010), pp. 4210–4229.

[154] Asad Ali Shahid, Loris Roveda, Dario Piga, and Francesco Braghin. “Learning Continuous Control Actions for Robotic Grasping with Reinforcement Learning”. In: *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2020, pp. 4066–4072.

[155] Jonah Siekmann, Kevin Green, John Warila, Alan Fern, and Jonathan Hurst. “Blind Bipedal Stair Traversal via Sim-to-Real Reinforcement Learning”. In: *arXiv preprint: 2105.08328* (2021).

[156] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, *et al.* “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587 (2016), pp. 484–489.

[157] Sourceforge. *SimSpark*. `http://simspark.sourceforge.net/`. Online. 2020.

[158] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[159] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. “Intriguing properties of neural networks”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. 2014.

[160] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. “Sim-to-Real: Learning Agile Locomotion For Quadruped Robots”. In: *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania, 2018, pp. 10–20.

[161] Matthew E Taylor and Peter Stone. “Transfer learning for reinforcement learning domains: A survey”. In: *Journal of Machine Learning Research* 10.Jul (2009), pp. 1633–1685.

[162] R. Tedrake, T. W. Zhang, and H. S. Seung. “Learning to walk in 20 minutes”. In: *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*. Vol. 95585. Beijing. 2005, pp. 1939–1412.

[163] Tamas Tettamanti, Tamas Luspay, Balazs Kulcsar, Tamas Peni, and Istvan Varga. “Robust Control for Urban Road Traffic Networks”. In: *IEEE Transactions on Intelligent Transportation Systems* 15.1 (2014), pp. 385–398.

[164] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. “Domain randomization for transferring deep neural networks from simulation to the real world”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 23–30.

[165] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033.

[166] Sergey Tomin, Gianluca Geloni, Igor Zagorodnov, Adam Egger, William Colocho, Alexander Valentinov, Yevgeniy Fomin, Ilya Agapov, Tyler Cope, Daniel Ratner, *et al.* “Progress in Automatic Software-based Optimization of Accelerator Performance”. In: *IPAC* (2016).

[167] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. “Congested traffic states in empirical observations and microscopic simulations”. In: *Phys. Rev. E* 62 (2 2000), pp. 1805–1824.

[168] Stephen Tu and Benjamin Recht. “Least-squares temporal difference learning for the linear quadratic regulator”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 5005–5014.

[169] William Uther and Manuela Veloso. *Adversarial reinforcement learning*. Tech. rep. Technical report, Carnegie Mellon University, 1997. Unpublished, 1997.

[170] Jeroen Van Baar, Alan Sullivan, Radu Cordorel, Devesh Jha, Diego Romeres, and Daniel Nikovski. "Sim-to-real transfer learning using robustified controllers in robotic tasks involving complex dynamics". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 6001–6007.

[171] Hado Van Hasselt. "Reinforcement learning in continuous state and action spaces". In: *Reinforcement learning*. Springer, 2012, pp. 207–251.

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

[173] Alexander Vandesompele, Gabriel Urbain, Hossain Mahmud, *et al.* "Body randomization reduces the sim-to-real gap for compliant quadruped locomotion". In: *Frontiers in neurorobotics* 13 (2019).

[174] Mel Vecerik, Oleg Sushkov, David Barker, Thomas Rothörl, Todd Hester, and Jon Scholz. "A practical approach to insertion with variable socket position using deep reinforcement learning". In: *2019 international conference on robotics and automation (ICRA)*. IEEE. 2019, pp. 754–760.

[175] Matej Večerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards". In: *arXiv preprint: 1707.08817* (2017).

[176] Joannès Vermorel and Mehryar Mohri. "Multi-armed Bandit Algorithms and Empirical Evaluation". In: *Machine Learning: ECML 2005*. Ed. by João Gama, Rui Camacho, Pavel B. Brazdil, Alípio Mário Jorge, and Luís Torgo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 437–448.

[177] Eugene Vinitsky, Aboudy Kreidieh, Luc Le Flem, Nishant Kheterpal, Kathy Jang, Cathy Wu, Fangyu Wu, Richard Liaw, Eric Liang, and Alexandre M Bayen. “Benchmarks for reinforcement learning in mixed-autonomy traffic”. In: *Conference on Robot Learning*. PMLR. 2018, pp. 399–409.

[178] Eugene Vinitsky, Nathan Lichtle, Kanaad Parvate, and Alexandre Bayen. *Optimizing Mixed Autonomy Traffic Flow With Decentralized Autonomous Vehicles and Multi-Agent RL*. 2020. arXiv: `2011.00120 [eess.SY]`.

[179] Jack M Wang, David J Fleet, and Aaron Hertzmann. “Optimizing walking controllers for uncertain inputs and environments”. In: *ACM Transactions on Graphics (TOG)* 29.4 (2010), p. 73.

[180] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[181] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. “IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 2496–2505.

[182] Marco A Wiering. “Multi-agent reinforcement learning for traffic light control”. In: *Machine Learning: Proceedings of the Seventeenth International Conference (ICML’2000)*. 2000, pp. 1151–1158.

[183] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[184] Markus Wulfmeier, Ingmar Posner, and Pieter Abbeel. “Mutual Alignment Transfer Learning”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, Nov. 2017, pp. 281–290.

[185] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. “Torcs, the open racing car simulator”. In: *Software available at http://torcs. sourceforge. net* 4.6 (2000).

[186] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese. “Gibson Env: Real-World Perception for Embodied Agents”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9068–9079.

[187] Li Hua Yu. “Generation of intense UV radiation by subharmonically seeded single-pass free-electron lasers”. In: *Physical Review A* 44.8 (1991), p. 5178.

[188] Weihao Yuan, Kaiyu Hang, Danica Kragic, Michael Y Wang, and Johannes A Stork. “End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer”. In: *Robotics and Autonomous Systems* 119 (2019), pp. 119–134.

[189] Yue Zhang, Christos G Cassandras, Wei Li, and Pieter J Mosterman. “A discrete-event and hybrid traffic simulation model based on SimEvents for intelligent transportation system analysis in Mcity”. In: *Discrete Event Dynamic Systems* 29.3 (2019), pp. 265–295.

[190] T. Zhao, H. Hachiya, G. Niu, and M. Sugiyama. “Analysis and improvement of policy gradient estimation”. In: *Advances in Neural Information Processing Systems*. 2011, pp. 262–270.

[191] W. Zhao, J. P. Queralta, and T. Westerlund. “Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey”. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2020, pp. 737–744.

[192] Y. Zhu, Z. Wang, C. Chen, and D. Dong. “Rule-Based Reinforcement Learning for Efficient Robot Navigation with Space Reduction”. In: *IEEE/ASME Transactions on Mechatronics* (2021), pp. 1–1.

About the author

Erica Salvato was born un Messina (Italy) on March 12th 1991. She received the B.Sc. degree in Electronic Engineering in 2015 from the University of Messina, and the M.Sc. degree in Electrical and Control Systems Engineering in 2018 from the University of Trieste (Italy). She is currently a PhD student of the Department of Engineering and Architecture at the University of Trieste. Her research focuses on the Artificial Intelligence application as a systems control tool but includes also Control Theory, Machine Learning, and Robotics.

List of Publications

Journal Publications

Francesca Cairoli, Gianfranco Fenu, Felice Andrea Pellegrino, and **Erica Salvato**. "Model Predictive Control of Glucose Concentration Based on Signal Temporal Logic Specifications with Unknown-Meals Occurrence". In: *Cybernetics and Systems* 51.4 (2020), pp. 426–441

Niky Bruchon, Gianfranco Fenu, Giulio Gaio, Marco Lonza, Finn Henry O'Shea, Felice Andrea Pellegrino, and **Erica Salvato**. "Basic reinforcement learning techniques to control the intensity of a seeded free-electron laser". In: *Electronics* 9.5 (2020), p. 781

Vittorio Casagrande, Gianfranco Fenu, Felice Andrea Pellegrino, Gilberto Pin, **Erica Salvato**, and Davide Zorzenon. "Machine learning for computationally efficient electrical loads estimation in consumer washing machines". In: *Neural Computing and Applications* (2021), pp. 1–12

Niky Bruchon, Gianfranco Fenu, Giulio Gaio, Simon Hirlander, Marco Lonza, Felice Andrea Pellegrino, and **Erica Salvato**. "An Online Iterative Linear Quadratic Approach for a Satisfactory Working Point Attainment at FERMI". in: *Information* 12.7 (2021), p. 262

Erica Salvato, Gianfranco Fenu, Eric Medvet, and Felice Andrea Pellegrino. "Crossing the Reality Gap: a Survey on Sim-to-Real Transferability of Robot Controllers in Reinforcement Learning". In: (2021)

Conference Publications

Francesca Cairoli, Gianfranco Fenu, Felice Andrea Pellegrino, and **Erica Salvato**. "Model predictive control of glucose concentration based on signal temporal logic specifications". In: *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*. IEEE. 2019, pp. 714–719

Niky Bruchon, Gianfranco Fenu, Giulio Gaio, Marco Lonza, Felice Andrea Pellegrino, and **Erica Salvato**. "Toward the Application of Reinforcement Learning to the Intensity Control of a Seeded Free-Electron Laser". In: *2019 23rd International Conference on Mechatronics Technology (ICMT)*. ed. by Adolfo Senatore and Truong Q. Dinh. Salerno: IEEE, Oct. 2019, pp. 1–6

Alexander Babichev, Vittorio Casagrande, Luca Della Schiava, Gianfranco Fenu, Imola Fodor, Enrico Marson, Felice Andrea Pellegrino, Gilberto Pin, **Erica Salvato**, Michele Toppano, *et al.* "Loads Estimation using Deep Learning Techniques in Consumer Washing Machines." In: *ICPRAM*. 2020, pp. 425–432

Erica Salvato, Arnob Ghosh, Gianfranco Fenu, and Thomas Parisini. *Control of a Mixed Autonomy Signalised Urban Intersection: An Action-Delayed Reinforcement Learning Approach*. 2021

Erica Salvato, Gianfranco Fenu, Eric Medvet, and Felice Andrea Pellegrino. "Characterization of Modeling Errors Affecting Performances of a Robotics Deep Reinforcement Learning Controller in a Sim-to-Real Transfer". In: *2021 44th International Convention on*

Information, Communication and Electronic Technology (MIPRO). IEEE. 2021, pp. 1154–1159

Acknowledgements

Achieving this goal would not have been possible without the contribution, the support and the backing of the people who been part of this journey, and those I had the pleasure of meeting during it.

First of all, I would like to thank the professional team that steered me through my PhD: my supervisor Felice Andrea Pellegrino, my co-supervisor Eric Medvet, Gianfranco Fenu, and Thomas Parisini. Their efforts, which transcend mere supervision, were essential to achieving this ultimate goal. I am truly grateful to them for what they have taught me and all the opportunities they have given me so far.

I wish to further spend some words to better acknowledge Felice who is more than a supervisor for me. Indeed, he has supported me all along the journey showing me his trust and respect. He has always advised me without influencing my decisions, and he has never failed to support me even when my choices were not in line with his own. From him I learned abnegation for work, but also respect for our own time. I learned that we can't always know the exact answer to a problem, but that if we work hard we can find it, and, moreover, that teamwork, done with passion and in a healthy environment, leads to the success of the individual, but also of the group. Thank you very much.

I can't miss thanking the *lunch-lab* guys: those who have always been there, and those who are just passing through. You have really made my work environment peaceful, by creating a mini family always ready to support and help each other. Nicolino, Giorgia, Eric, Andrea, Luca, Claudia, Natalia, and Viola thank you in particular!

Some few words for Eric my co-supervisor, but also a friend. Thanks for all the walks (sometimes tiring), and conversations. Thanks for all the time that you listen to my paturnias by advising me, or teasing me properly. The family environment that has been created is primarily thanks to you.

Thank to my best friend, for me like a sister, Giulia. She followed me, advised me, and borne me....as always! After all, “every brunette has her blonde!"

My biggest thank you is for my family who have supported, sustained, and encouraged me throughout this course. They have always believed in my abilities, which is something that I often struggle to do on my own. Covid has led us to live at greater distances than normal, but family is always family. Thank you mom, dad, Simona and Chiara. In particular, I wish my two sisters success in all their dreams, and remember that a job done with love leads to excellent results.

Last but not least thank you Emanuele, my love, friend, and guideline. You shared everything with me, never failing to support me. We have been through a lot together.... even a pandemic. You are, and will always be, my most important person, even with this “marina-di-cannuccing" ideas. I love you.

Erica