IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 8, 2021, accepted October 29, 2021, date of publication November 8, 2021,
date of current version November 19, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3126658

Crossing the Reality Gap: A Survey on Sim-to-Real Transferability of Robot Controllers in Reinforcement Learning

ERICA SALVATO, GIANFRANCO FENU, ERIC MEDVET, AND FELICE ANDREA PELLEGRINO, (Member, IEEE)
Department of Engineering and Architecture, University of Trieste, 34127 Trieste, Italy

Corresponding author: Erica Salvato (erica.salvato@phd.units.it)

This work was supported in part by the Italian Ministry for Research in the framework of the 2017 Program for Research Projects of National Interest (PRIN), under Grant 2017YKXYXJ.

ABSTRACT 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. However, the *trial and error* intrinsic nature of RL may result in long training time on real robots and, moreover, it may lead to dangerous outcomes. While simulators are useful tools to accelerate RL training and to ensure safety, they often are provided only with an approximated model of robot dynamics and of its interaction with the surrounding environment, thus resulting in what is called the *reality gap* (RG): a mismatch of simulated and real control-law performances caused by the inaccurate representation of the real environment in simulation. The most undesirable result occurs when the controller learnt in simulation fails the task on the real robot, thus resulting in an unsuccessful *sim-to-real* transfer. The goal of the present survey is threefold: (1) to identify the main approaches to face the RG problem in the context of robot control with RL, (2) to point out their shortcomings, and (3) to outline new potential research areas.

INDEX TERMS Reality gap, reinforcement learning, robotics, sim-to-real.

I. INTRODUCTION

Reinforcement Learning (RL) [1] allows to design controllers (often referred to as *agents*) with the capability of learning an optimal behaviour by interacting with the *environment*. The behaviour is defined in terms of state-action pairs, also known as *policy*, which is learnt through a *trial and error* process.

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 [1], 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 [1]. 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 [2], [3]. For this purpose, approximators of the policy or of some supporting element, such as value function, or of both, are required [1]. 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.) [4]–[6]. 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 behaviour as closely as possible. In principle, simulators allow to train the controller with faster and safer procedures: once the policy has been learnt, it is transferred to the real system (*sim-to-real* transfer) [7]. However, sim-to-real transfer is only effective when the simulator is given a sufficiently accurate model of the real robot and the environment [8]; unfortunately, the more accurate the

The associate editor coordinating the review of this manuscript and approving it for publication was Yongqiang Cheng.

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 learnt on simulator degrades once applied on the real world is the so-called *reality gap* (RG) [9]. 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 [10]. Indeed, several works faced the RG problem in other frameworks, such as Evolutionary Computation [9], [11]–[17] or Model Predictive Control [18]–[21].

However, here we focus our attention only on those works facing the RG problem on robot controllers learnt 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) generalisation 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 [22], here we conduct a more in-depth analysis. We introduce the main concept behind the RG in a general RL framework and we survey relevant and recent literature concerning RG in the context of robot control with RL. According to our analysis, the approaches for coping with RG in this context fall into three broad categories: domain randomisation (DR), adversarial reinforcement learning (ARL), and transfer learning (TL). Hence, we introduce a general RL framework suitable to be specialised for each of the mentioned approaches.

The aims of the present work are:

(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;
(3) to identify new possible research areas.

The remainder of the paper is organised as follows. In Section II we describe a formal framework for RL useful to better explain the key elements of DR, ARL, and TL. In Section III we introduce RL as a means for robot control and discuss the RG in the context of robotics. In Section IV we survey and discuss the current methodologies to face the RG, relying, where possible, on the provided formalism. Finally, in Section V, we draw the conclusions.

II. REINFORCEMENT LEARNING

Reinforcement Learning (RL) can be employed to perform optimal data-driven control without the need to rely on a mathematical model of system dynamics [1], [23], [24].

It is typically described as a Markov Decision Process (MDP); i.e., a tuple (X, A, p, r) defined by the state set X, the control set A, the transition probability p, and the immediate reward $r \in \mathbb{R}$. In brief, it expresses a discrete-time stochastic control process in which a scalar r 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 [25]. However, what follows also applies to MDP settings by defining some of the following elements in terms of expected values.

A. 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 $a^{(0)}, a^{(1)}, \ldots$, evolves according to the following laws:

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

$$o^{(k+1)} = g\left(x^{(k+1)}\right) = g\left(f\left(x^{(k)}, a^{(k)}\right)\right). \tag{2}$$

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 $a^{(0)}, a^{(1)}, \ldots$, evolves according to Equations (1) and (2) and:

$$r^{(k+1)} = h\left(x^{(k)}, a^{(k)}\right). \tag{3}$$

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 amount to employing a Partially Observable Markov Decision Process (POMDP), which requires to introduce the concept of *belief* [26]. 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 $\pi : O \to A$.

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

$$\begin{aligned} \pi^* &= \arg\max_{\pi \in \Pi} \sum_{k=0}^{+\infty} \gamma^k h\left(x^{(k)}, \pi\left(o^{(k)}\right)\right) \\ &= \arg\max_{\pi \in \Pi} J_\pi\left(x^{(0)}\right), \end{aligned} \tag{4}$$

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 with $\pi \leftrightarrow E$ the closed-loop system where π determines the control input to be applied on E, represented

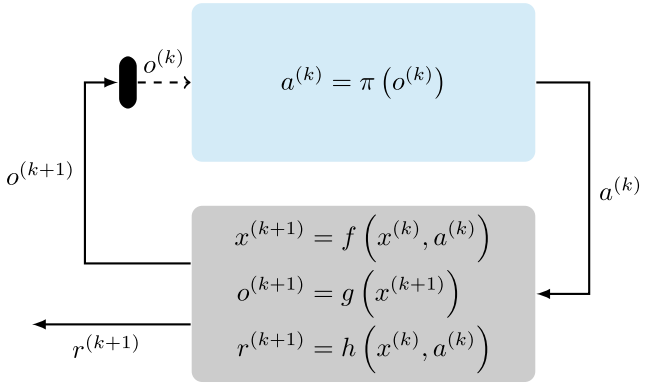


FIGURE 1. The closed-loop system $\pi \leftrightarrow E$ where a policy (light blue block above) applies a control input $a^{(k)}$ to an environment (gray block below) that outputs an observation $o^{(k+1)}$ and a reward $r^{(k+1)}$.

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

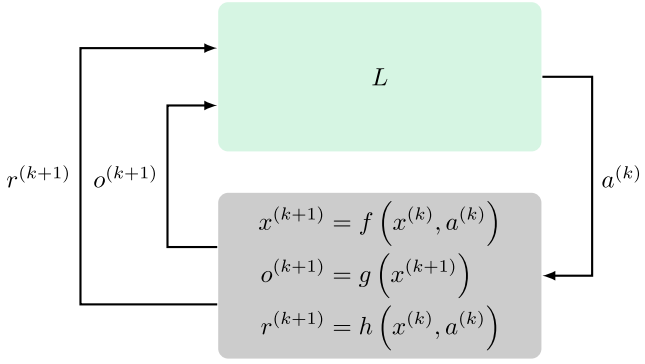


FIGURE 2. The closed-loop system $L \leftrightarrow E$ where a RL-based agent L (green block above) applies a control input $a^{(k)}$ to an environment (gray block below) that outputs an observation $o^{(k+1)}$ and a reward $r^{(k+1)}$.

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 selected control inputs. Its global goal is to learn a controller π^* in E which satisfies (4).

The learning procedure can either be *episodic* or in a single chapter (*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 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 categorised in three main categories [1]:

- *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 $x^{(k)}$ by applying π and the cumulative reward obtained from $x^{(k)}$ by first choosing an $a^{(k)}$ and then applying π. The optimal policy corresponds to the optimal V or Q functions:

$$V^*(x^{(k)}) = \max_\pi J_\pi\left(x^{(k)}\right) \tag{5}$$

$$Q^*(x^{(k)}, a^{(k)}) = \max_\pi h\left(x^{(k)}, a^{(k)}\right) + \gamma J_\pi\left(x^{(k+1)}\right). \tag{6}$$

  In brief, by interacting with the environment and observing rewards, either V^* or Q^* are estimated, thus leading to the optimal policy.
- *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 maximise J_{π_θ}, either employing gradient-based or gradient-free optimisation techniques [27].
- *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 optimisation (Actor).

Further technicalities are not necessary for the purpose of this work. For additional details on RL we refer readers to [1], [25], [28], [29].

III. REINFORCEMENT LEARNING IN ROBOTICS

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, as a matter of fact, the most effective methods to date are model-based [30]–[32]. In addition, policy search approaches result in more efficient trainings in terms of time needed for convergence [33]–[38]. 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 1, we report the tasks that are more

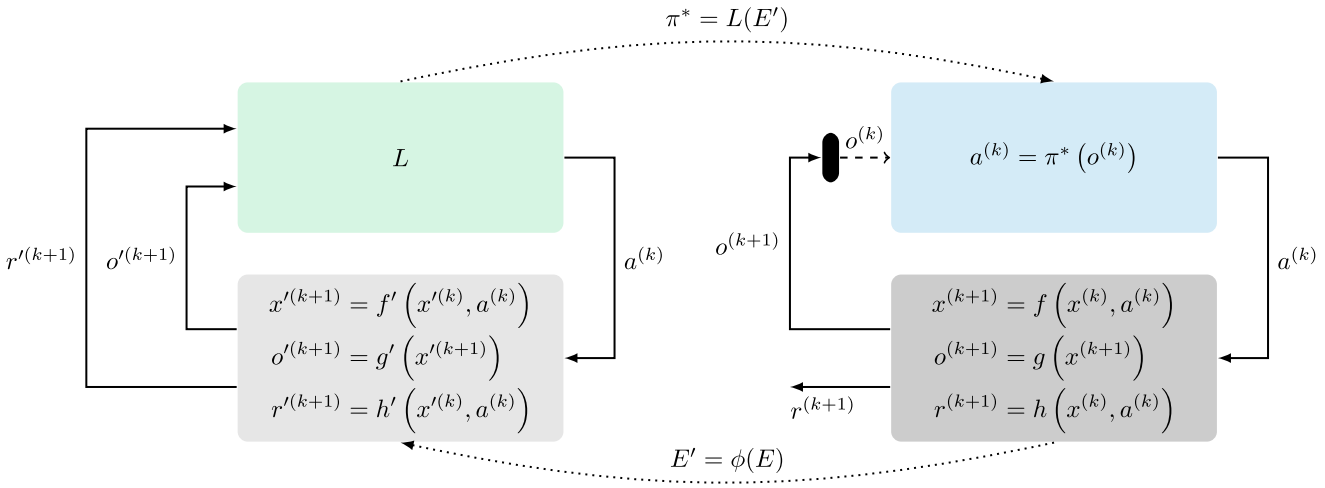


FIGURE 3. Schematic representation of the application of RL to the robot control problem using simulation. First, given the real robot and it environment E (block at bottom right), a simulator E' (block at bottom left) is obtained by modelling E using ϕ. Then an agent L (block at top left) learns a policy π^* in simulation. Finally, the learnt policy is transferred to the real robot where it can be deployed (block at top right).

TABLE 1. Overview of the robotic tasks involving RL controllers.

Tasks	References
Navigation	[39], [40], [41], [42]
Manipulation	[43], [44], [45], [46], [47], [3], [8], [48]
Motion control	[49], [50], [51], [52], [53], [54], [55], [56], [57], [58], [59]
Locomotion	[49], [50], [51], [60], [61], [62]

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.

When tackling a robot control problem with RL, according to the formulation and notation above provided, 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 [4], [63].

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 modelling a real system described by E as a simulated system described by E' such that a policy learnt 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 3, 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 learnt in simulation often exhibits

[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.

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 learnt 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 learnt policy π^* is said to be non-transferable.

Note that, although it is not always clearly emphasised, a mandatory step ahead of the sim-to-real transfer is to perform a test of the learnt policy on the simulator itself.[2] Otherwise, there is no guarantee of the learnt 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 learnt 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 behaviour 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 resulting o and o' are different; or, possibly, E and E' differences in h and h' lead to two different optimisation 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 behaviour 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 summarises 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 III. Once that the III has been executed, (4) we compare the performance obtained in the π tests on E and E'.

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

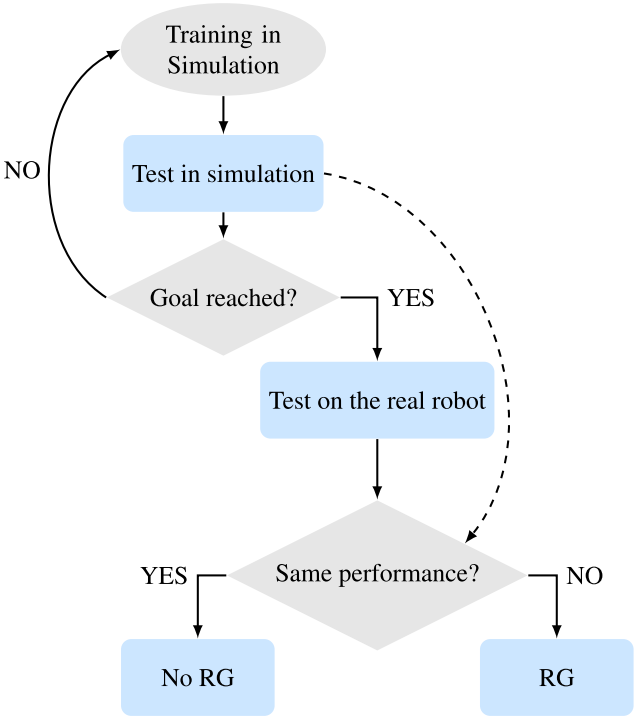


FIGURE 4. Flowchart of the routine to test the presence of the RG.

IV. METHODOLOGIES FOR SOLVING RG

In the present paper, 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 randomisation (DR), adversarial RL (ARL), transfer learning (TL). All the articles discussed below are summarised in Table 2 according to this categorisation. 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 5 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 Section II, which allows to understand each approach.

A. DOMAIN RANDOMISATION

Domain randomisation (DR) has already achieved good results in sim-to-real transfer of robotics controllers outside RL [54], [75]–[79]. The main idea behind this approach is what in control theory is called *robust control under either parametric or non parametric uncertainty* [80], [81], that is

TABLE 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
[49]	TL	✓	✓	MuJoCo [64]
[39]	DR	✓	✓	Blender [65]
[43]	TL		✓	MuJoCo
[50]	DR	✓		MuJoCo
[51]	ARL	✓		MuJoCo
[60]	TL	✓	✓	SimSpark [66], Gazebo [67]
[52]	TL	✓		MuJoCo, DART [68]
[53]	DR	✓		Vortex [69], Bullet[70]
[44]	DR		✓	MuJoCo
[45]	DR		✓	PyBullet [71]
[54]	DR	✓	✓	PyBullet
[55]	TL	✓	✓	MuJoCo, PyBullet
[56]	DR		✓	MuJoCo
[40]	TL		✓	Gibson [72]
[57]	DR	✓	✓	MuJoCo
[58]	ARL	✓		TORCS [73]
[46]	DR		✓	PyBullet
[47]	TL		✓	Gazebo
[61]	DR	✓	✓	MuJoCo
[41]	DR	✓	✓	Gazebo

the design of controllers able to guarantee certain properties despite some tolerable parameters variations and/or noise.

We call $\tilde{E}' = \tilde{\phi}(E)$ a corrupted simulator described by a tuple $(\tilde{X}', \tilde{A}', \tilde{O}', Z', \Upsilon', \tilde{f}'_{\xi'}, \tilde{g}'_{\psi'}, h)$ 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 $\tilde{a}'^{(0)}, \tilde{a}'^{(1)}, \ldots$, a process disturbance sequence $\zeta'^{(0)}, \zeta'^{(1)}, \ldots$, and a measurement disturbance sequence $\upsilon'^{(0)}, \upsilon'^{(1)}, \ldots$, a corrupted simulator $\tilde{E}'$ evolves according to:

$$\tilde{x}'^{(k+1)} = \tilde{f}'_{\xi'}\left(\tilde{x}'^{(k)}, \tilde{a}'^{(k)}, \zeta'^{(k)} | \xi'\right) \tag{7}$$

$$\tilde{o}'^{(k+1)} = \tilde{g}'_{\psi'}\left(\tilde{x}'^{(k+1)}, \upsilon'^{(k)} | \psi'\right) \tag{8}$$

$$\tilde{r}'^{(k+1)} = h\left(\tilde{x}'^{(k)}, \tilde{a}'^{(k)}\right). \tag{9}$$

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 (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.

TABLE 3. List of the surveyed articles that apply the DR approach.

Ref.	Algorithm	Tasks
[39]	DQN[h]	Vision-based flight
[50]	TRPO[a]	Inverted pendulum, Half cheetah [82], Hopper [82], Walker [82]
[53]	TRPO[a]	Ball on plate with robotic arm
[44]	HER[b]+RDPG[c]	Pushing task with robotic arm
[45]	DDPGfD[f]	Deformable object manipulation
[54]	PPO[e]	Trotting and galloping of quadruped
[56]	A3C[d]	Marble maze game with robotic arm
[57]	PPO[e]	Rubik's cube with robotic hand
[46]	QT-Opt[g]	Rvision-based control task of a robotic arm for grasping
[61]	PPO	Climb and descend stairs with bipedal robot
[41]	A3C	Wheeled mobile platform navigation

[a]Trust Region Policy Optimisation [83], [b]Hindsight Experience Replay [84], [c]Recurrent Deterministic Policy Gradient [85], [d]Asynchronous Actor-Critic Agents [86], [e]Proximal Policy Optimization [87], [f]Deep Deterministic Policy Gradient from Demonstration [88]. [g]Q-function Targets via Optimization [89] [h]Deep Q Network [90]

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 6 graphically summarises this process. The resulting controller π^*, learnt by maximising 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 a π^* affected by a tolerable RG, the result is a sim-to-real transferable controller.

Table 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 2); instead the controller robustness with respect to the perturbations is tested. We discuss each of the paper below.

In Sadeghi and Levine [39] 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

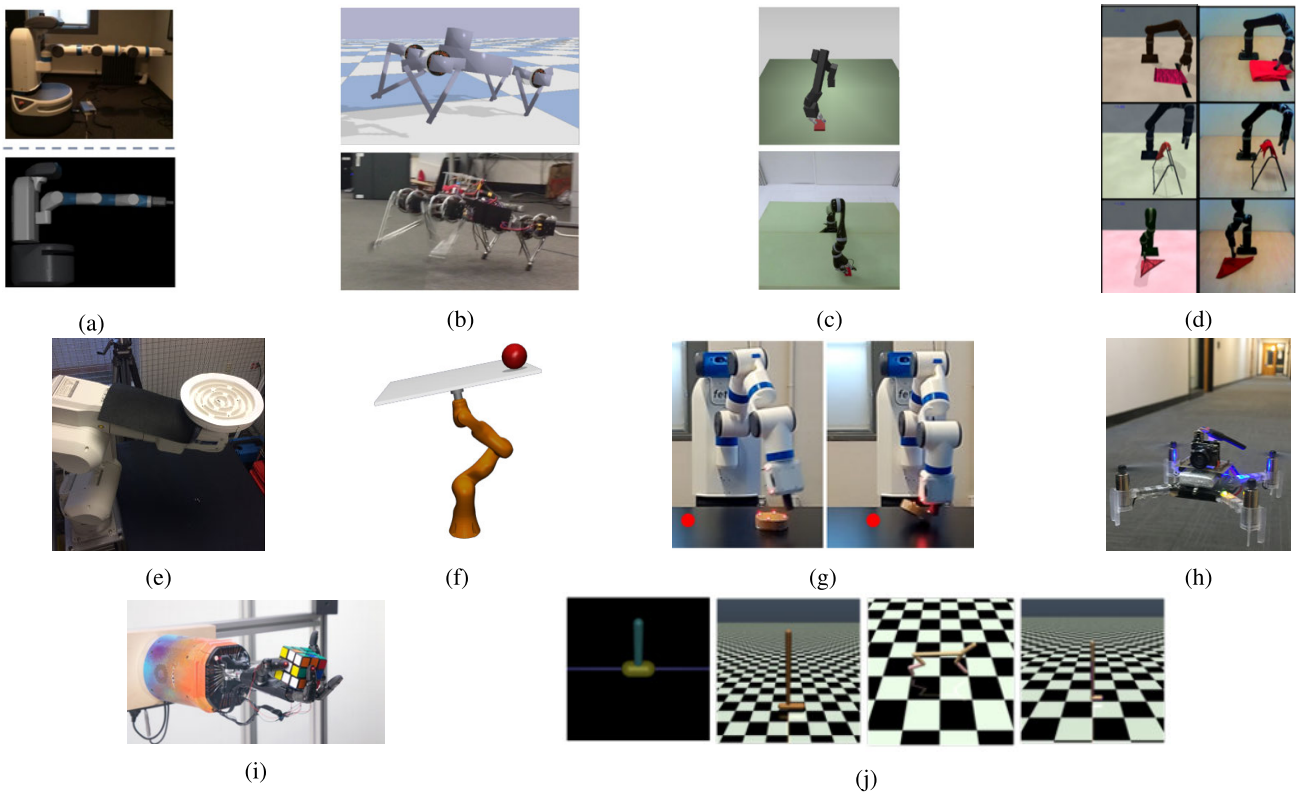


FIGURE 5. Overview of some robotic tasks considered in the surveyed articles to address the RG. (a) The Fetch robot used in [49]. (b) The Minitaur of [54]. (c) The robot employed for manipulation task in [74]. (d) The robotic arm engaged in deformable object manipulation of [45]. (e) The Marble maze game of [56]. (f) The ball on plate system used by [53]. (g) The Fetch robot used for the pushing task of [44]. (h) The quadrotor employed for the autonomous navigation task of [40]. (i) The five-finger humanoid hand used in [57]. (j) The classical Open AI Gym environment used to test in simulation several strategies [49]–[52].

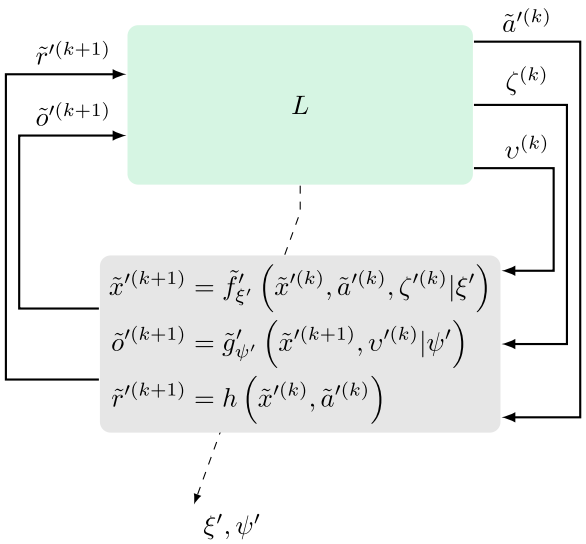


FIGURE 6. Schematic representation of $L \leftrightarrow \tilde{E}'$ interaction in DR approaches.

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 learnt 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 randomisation enacts a wide variety of environments, and shows very good performance during 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.* [50] 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 [91], in which ξ', $\upsilon'^{(k)}$, and $\zeta'^{(k)}$ alternately assume the form of isometrically scaled versions of Fast Gradient Sign Method (FGSM) [92]. Here, the controller is parametrized by θ (π_θ) and updated following the on-policy vanilla Trust Region Policy Optimisation (TRPO) [83]. 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 generalise well [93].

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 Optimisation with Transferability Assessment (SPOTA) algorithm, designed in [53], uses randomised physics parameters, drawn from a probability distribution $\xi' \sim \rho_\kappa(\xi')$ (parametrized by κ), to perform a robust optimisation 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 Optimisation Bias (SOB) concept: an error caused by an optimistic bias of the optimisation procedure, whose existence has been proven by [94]. 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 Optimisation 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 *et al.* [44] shows the effectiveness of memory-based policies (i.e., policies learnt by using past memory for future learning [95]–[97]) to deal with the RG, introducing DR to generalise environment dynamics. Hindsight Experience Replay [84] has been used for the purpose: a technique able to generalise over different goals using past experience as a baseline. In this case, the parameters ξ', the measurement noise $\upsilon'^{(k)}$, 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, learnt 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 [88] is adopted in [45] 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 randomisation 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 randomisation can lead to unsuccessful transfers.

Controller sim-to-real transferability has also been tested on locomotion tasks of a Minitaur quadruped of Ghost Robotics [54]. Here authors have used Proximal Policy Optimisation (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 randomised $\zeta'^{(k)}$, $\upsilon'^{(k)}$, 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 shown that controllers learnt in simulation with large observation space lead to bad results when transferred to real robot.

Van Baar *et al.* [56] shows the benefits of using DR and Asynchronous Actor Critic Agent (A3C) algorithm [86] (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 analysed 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 learnt 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 [57], Automatic Domain Randomisation (ADR) is proposed in order to transfer a policy learnt 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 Randomised-to-Canonical Adaptation Network (RCAN) is conversely introduced in [46]. 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 learnt by using an image-conditioned Generative Adversarial Network (cGAN) [98], able to map an image of a domain D into an adapted image of the canonical domain D_c; i.e., $G : D \rightarrow D_c$. The resulting image of D_c is then sent to the controller which is learnt by using Q-function Targets via Optimisation (QT-Opt) [89]. During training, cGAN receives randomised 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.* [61] 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.* [41] 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 randomisation 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, hypothesising 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.

B. 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 randomisation: 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 controller (the adversarial) to generate more difficult models to handle (those that minimise the cumulative reward). Figure 7 graphically summarises the process of ARL.

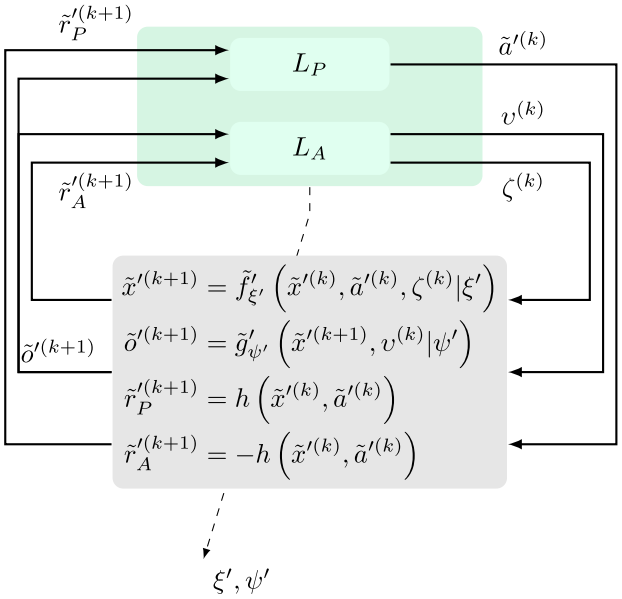


FIGURE 7. Schematic representation of $L \leftrightarrow \tilde{E}'$ interaction in ARL approaches.

Given a corrupted simulator $\tilde{E}'$ of E, defined by a tuple $(\tilde{X}', \tilde{A}', \tilde{O}', Z', \Upsilon', \tilde{f}'_{\xi'}, \tilde{g}'_{\psi'}, h)$ and evolving according to Equations (7) to (9),, L_P and L_A interact with $\tilde{E}'$ seeking to maximise their respective discounted cumulative reward [99].

We denote with $\tilde{r}_P'^{(k+1)}$ the reward of L_P. A common choice [100] is to provide L_A with a reward $\tilde{r}_A'^{(k+1)} = -\tilde{r}_P'^{(k+1)}$. As a result two controllers are learnt:

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

$$\max_{\pi_P} \min_{\pi_A} J_{\pi_P,\pi_A}\left(\tilde{x}'^{(0)}\right), \tag{10}$$

falling in a *worst-case* approach that in control theory is known as *min-max optimal control* (also referred to H_∞-control) [101]–[103].

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:

$$\pi_L^* = \arg\max_{\pi_P} \arg\min_{\pi_A} J_{\pi_P,\pi_A}. \tag{11}$$

Table 4 summarises the articles that use ARL for learning robust controllers, along with the respective learning algorithms and the considered tasks.

TABLE 4. List of the surveyed articles that apply the ARL approach.

Ref.	Algorithm	Tasks
[51]	TRPO[a]	Inverted pendulum, Half cheetah, Swimmer [82], Hopper, Walker 2D, Ant [82]
[58]	Ensemble DQN[b]	Autonomous driving

[a]Trust Region Policy Optimization [83], [b]Ensemble Deep-Q Network [104].

The ARL approach was firstly introduced by [51], 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 destabilising disturbances to environment (in particular only ξ' disturbances). The proposed solution can be summarised 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 [82]. In a first experiment, the authors compare mean and variance of the cumulative reward over 50 RARL policies, obtained using different seeds and initialisation, with TRPO ones. For all tasks RARL behaves better than TRPO in terms of mean and variance. In a second experiment, [51] 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.* [58] 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 maximise 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.

C. TRANSFER LEARNING

The previously discussed approaches are aimed at controllers that, once learnt 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), penalising 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 generalisation is possible not only *within* task but also *between* tasks [105]. Therefore, since a task can be entirely defined by an environment, considering a second different but *compatible* environment,[3] the controller learnt for the first is expected to be a helpful tool to speed-up the second learning process, whether or not it involves the same agent. Thereby, in RL, in which the controller is the result of a trial and error process, TL could be employed

[3]Two environments $E = (X, A, O, f, g, h)$ and $E' = (X', A', O', f', g', h')$ are compatible if and only if $A = A'$ and $O = O'$. Thus, 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'.

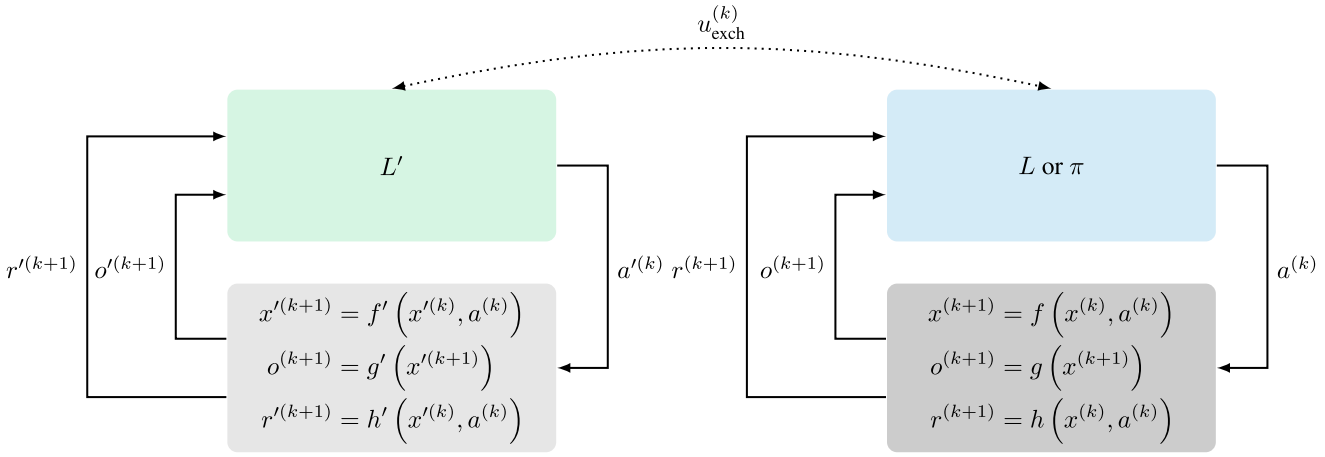


FIGURE 8. Schematic representation of TL approaches.

TABLE 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
[49]	TRPO[a]	$a'(k)$	C-TL	2D-TL	IDNN[e]	Reacher, Half cheetah, Hopper, Humanoid, Fetch robot, Trajectory control
[43]	A3C[c]	NN activation functions	1-TL	1D-TL	RL	Jaco robot manipulation
[60]	CMA-ES[b]	$a(k), a'(k)$	C-TL	2D-TL	RL	Humanoid bipedal locomotion
[52]	TRPO[a]	$o'^{(k)}, a'^{(k)}, o^{(k)}, a^{(k)}$	C-TL	2D-TL	RL	Reacher, Hopper 2D, Reacher 2D
[55]	PPO[g]	π'^{*}	1-TL	1D-TL	RL	Pusher, Striker, Ergo Reacher, ErgoShield
[40]	DQL[d]	NN weights	1-TL	1D-TL	MPC[f]	Quadrotor collision avoiding
[47]	DQL[d]	$O', Q(x'^{(k)}, a'^{(k)})$	1-TL	1D-TL	RL	Robot object manipulation

[a]Trust Region Policy Optimization [83], [b]Covariance Matrix Adaptation-Evolutionary Strategy [106], [c]Asynchronous Actor-Critic Agents [86], [d]Double Q-Learning [107], [e]Inverse Dynamics Neural Network, [f] Model Predictive Control [108]

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 learnt 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 will 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 8).

We categorise 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 5 summarises the articles that use TL and characterises them in terms of these four factors. The table also shows the tasks and algorithms.

Christiano *et al.* [49] adopted TL to perform sim-to-real control input adaptation. They assume that if simulator does not exactly replicate the real robot behaviour, 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 learnt 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 $u^{(i)}_{\text{exch}} = [u'^{(i)}\ u^{(i)}]^T = [o'^{(i)}\ \tau_i]^T$. It is worth remarking that what is actually learnt in this case is an inverse dynamic model (implemented as a neural network) that must be employed in conjunction with the controller learnt 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 [60], 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 [109], 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 optimisation 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 optimisation problem is solved in order to find those E' parameters able to minimise 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 minimise J_{eval}. The optimal policy is the $\pi \in \Pi_c$ such that J_{eval} is minimised 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 $a'^{(k)} = m(a^{(k)})$, learnt 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 $u^{(k)}_{\text{exch}} = [u'^{(k)}\ u^{(k)}]^T = [a'^{(k)}, a^{(k)}]^T$. 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.* [52] proposed Mutual Alignment Transfer Learning (MATL), a method that relies on a Generative Adversarial Network (GAN) [110]. The main idea is similar to [49]: 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 maximise 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 $u^{(k)}_{\text{exch}}$ is the D output and it is equal to $[-\log(D(\tau_\kappa))\ \ \log(D(\tau_\kappa))]^T$. Wulfmeier *et al.* [52] evaluate their approach on various RL tasks: rllab [111], OpenAI Gym, and DartEnv [82]. 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 [43], where progressive nets [74] 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 *et al.* [43] 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, $u_{\text{exch}}^{(k)} = [u'^{(i)}\ 0]^T = [(\text{act}_{1,s}, \ldots, \text{act}_{k,s})\ 0]^T$. The authors evaluate their approach on a robot manipulation task on Jaco arm [112] 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 [55], 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 Optimisation algorithm is learnt 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 $u_{\text{exch}}^{(k)} = [u'^{(i)}\ 0]^T = [\pi\ 0]^T$. 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 [113] 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 [40], 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 minimise 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 maximises the expected discounted predicted reward within an horizon h. In this context, $u_{\text{exch}}^{(k)} = [u'^{(k)}\ 0]^T = [\theta_{\text{VP}}\ 0]^T$. As prescribed by the MPC approach, only the first control input is applied; hence, the process is repeated. Kang *et al.* [40] 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.* [47] 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 (learnt in step IV-C) 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 minimise 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 randomisation idea of [75]. Experimental results show that their approaches surpass the one of [75] in terms of performances and, in particular, (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.

D. 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 [114]. Here the authors propose to learn a policy in a randomised 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 [115], [116]. In Meta-RL, given a distribution over tasks, the agent learns an adaptive policy that maximises the expected reward for a new task from the distribution. A recent work [117] has shown the great ability of this approach to generalise 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 [118]. Here, a probabilistic model of the system dynamics is learnt incorporating uncertainty by using only some trial on the real system and a policy is learnt 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 [119], 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 [120].

V. CONCLUSION AND OPEN CHALLENGES

Training a RL robotic controller in real-time in its actual environment is a costly process. While simulators can alleviate the problem, the approximations often present in the employed models play a crucial role in determining the effectiveness of the learnt controllers. The more accurate the model of the robot (and the surrounding), the more effective the controller but, at the same time, the greater the computational cost of learning it. When performances of the controller in the simulator and in real robot are different, a RG exists and the controller is not sim-to-real transferable.

In the present article, we provided a formal framework for the RG and reviewed the most relevant existing methods aiming to achieve sim-to-real transferable controllers in robotics RL applications. We surveyed the literature concerning RL and RG and categorised the approaches as: domain randomisation, adversarial reinforcement learning, and transfer learning. Moreover, we described them in detail according to the proposed framework and in terms of the employed algorithms, involved tasks, and evaluation methods.

We conclude commenting on some significant open challenges. Each one of the examined approaches appears tailored to a specific task, and its applicability to other, potentially different, tasks is not clear. 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 a significant open problem is that of providing a proper index able to reveal and quantify the RG. Indeed, being able to characterise and quantify the RG would (i) enable systematic comparison among different techniques, hence favouring the advancement of research, and (ii) allow to use the measure of RG directly as an optimisation objective, hence putting the transferability as a direct goal in the learning of RG-aware controllers. Another significant open problem is the sample efficiency of the developed approaches. As highlighted in [121], RL is very data-intensive. The computation effort required for learning a RL agent involved in a complex task can be huge, thus limiting the practical applicability of such methods. The surveyed approaches lead to learning paradigms that unquestionably aggravate this issue, especially in those cases in which an interaction with different environment domains is involved.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[2] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1238–1274, 2013.
[3] A. A. Shahid, L. Roveda, D. Piga, and F. Braghin, "Learning continuous control actions for robotic grasping with reinforcement learning," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2020, pp. 4066–4072.
[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.
[5] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, vol. 3, Apr. 2004, pp. 2619–2624.
[6] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *Experimental Robotics IX*. Berlin, Germany: Springer, 2006, pp. 363–372.
[7] M. Breyer, F. Furrer, T. Novkovic, R. Siegwart, and J. Nieto, "Flexible robotic grasping with sim-to-real transfer based reinforcement learning," 2018, *arXiv:1803.04996*.
[8] E. Salvato, G. Fenu, E. Medvet, and F. A. Pellegrino, "Characterization of modeling errors affecting performances of a robotics deep reinforcement learning controller in a sim-to-real transfer," in *Proc. 44rd Int. Conv. Inf., Commun. Electron. Technol. (MIPRO)*, 2021, pp. 1324–1329.
[9] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Proc. Eur. Conf. Artif. Life*. Berlin, Germany: Springer, 1995, pp. 704–720.
[10] S. Hofer, K. Bekris, A. Handa, J. C. Gamboa, M. Mozifian, F. Golemo, C. Atkeson, D. Fox, K. Goldberg, J. Leonard, C. Karen Liu, J. Peters, S. Song, P. Welinder, and M. White, "Sim2Real in robotics and automation: Applications and challenges," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 2, pp. 398–400, Apr. 2021.

[11] J. B. Pollack, H. Lipson, S. Ficici, P. Funes, G. Hornby, and R. A. Watson, "Evolutionary techniques in physical robotics," in *Proc. Int. Conf. Evolvable Syst.* Berlin, Germany: Springer, 2000, pp. 175–186.

[12] S. Nolfi, D. Floreano, O. Miglino, and F. 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*. Cambridge, MA, USA: MIT Press, 1994, pp. 190–197.

[13] D. Floreano and J. Urzelai, "Evolution of plastic control networks," *Auto. robots*, vol. 11, no. 3, pp. 311–317, 2001.

[14] C. Hartland and N. Bredeche, "Evolutionary robotics, anticipation and the reality gap," in *Proc. IEEE Int. Conf. Robot. Biomimetics*, Dec. 2006, pp. 1640–1645.

[15] J.-B. Mouret and K. Chatzilygeroudis, "20 years of reality gap: A few thoughts about simulators in evolutionary robotics," in *Proc. Genetic Evol. Comput. Conf. Companion*, 2017, pp. 1121–1124.

[16] S. Koos, J.-B. Mouret, and S. Doncieux, "The transferability approach: Crossing the reality gap in evolutionary robotics," *IEEE Trans. Evol. Comput.*, vol. 17, no. 1, pp. 122–145, Feb. 2012.

[17] S. Kriegman, D. Blackiston, M. Levin, and J. Bongard, "A scalable pipeline for designing reconfigurable organisms," *Proc. Nat. Acad. Sci. USA*, vol. 117, no. 4, pp. 1853–1859, 2020. [Online]. Available: https://www.pnas.org/content/117/4/1853

[18] A. S. Badwe, R. D. Gudi, R. S. Patwardhan, S. L. Shah, and S. C. Patwardhan, "Detection of model-plant mismatch in MPC applications," *J. Process Control*, vol. 19, no. 8, pp. 1305–1313, 2009.

[19] M. Kano, Y. Shigi, S. Hasebe, and S. Ooyama, "Detection of significant model-plant mismatch from routine operation data of model predictive control system," *IFAC Proc. Volumes*, vol. 43, no. 5, pp. 685–690, 2010.

[20] S. Selvanathan and A. K. Tangirala, "Diagnosis of poor control loop performance due to model–plant mismatch," *Ind. Eng. Chem. Res.*, vol. 49, no. 9, pp. 4210–4229, May 2010.

[21] L. E. Olivier and I. K. Craig, "Model-plant mismatch detection and model update for a run-of-mine ore milling circuit under model predictive control," *J. Process Control*, vol. 23, no. 2, pp. 100–107, Feb. 2013.

[22] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," in *Proc. IEEE Symp. Comput. Intell. (SSCI)*, Dec. 2020, pp. 737–744.

[23] R. Bellman, "On the theory of dynamic programming," *Proc. Nat. Acad. Sci. USA*, vol. 38, no. 8, p. 716, 1952.

[24] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Boca Raton, FL, USA: CRC Press, 2017.

[25] D. P. Bertsekas, *Reinforcement Learning and Optimal Control*. Belmont, MA, USA: Athena Scientific, 2019.

[26] A. C. Rodriguez, R. Parr, and D. Koller, "Reinforcement learning using approximate belief states," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1036–1042.

[27] H. Van Hasselt, "Reinforcement learning in continuous state and action spaces," in *Reinforcement Learning*. Berlin, Germany: Springer, 2012, pp. 207–251.

[28] M. P. Deisenroth, G. Neumann, and J. Peters, *A Survey on Policy Search for Robotics*. Boston, MA, USA: Now, 2013.

[29] B. Recht, "A tour of reinforcement learning: The view from continuous control," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 2, no. 1, pp. 253–279, 2019.

[30] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *Proc. Int. Conf. Mach. Learn.*, vol. 97, 1997, pp. 12–20.

[31] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Proc. Adv. Neural Inf. Process. Syst.*, 2007, pp. 1–8.

[32] H. Durrant-Whyte, N. Roy, and P. Abbeel, *Learning to Control a Low-Cost Manipulator Using Data-Efficient Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 2012, pp. 57–64. [Online]. Available: https://ieeexplore.ieee.org/document/6301026

[33] V. Gullapalli, J. A. Franklin, and H. Benbrahim, "Acquiring robot skills via reinforcement learning," *IEEE Control Syst.*, vol. 14, no. 1, pp. 13–24, Feb. 1994.

[34] H. Miyamoto, S. Schaal, F. Gandolfo, H. Gomi, Y. Koike, R. Osu, E. Nakano, Y. Wada, and M. Kawato, "A kendama learning robot based on bi-directional theory," *Neural Netw.*, vol. 9, no. 8, pp. 1281–1302, 1996.

[35] R. Tedrake, T. W. Zhang, and H. S. Seung, "Learning to walk in 20 minutes," in *Proc. 14th Yale Workshop Adapt. Learn. Syst.*, Beijing, China, vol. 95585, 2005, pp. 1412–1939.

[36] J. Peters and S. Schaal, "Learning to control in operational space," *Int. J. Robot. Res.*, vol. 27, no. 2, pp. 197–212, 2008.

[37] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, nos. 7–9, pp. 1180–1190, Mar. 2008.

[38] 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," *IEEE Trans. Robot.*, vol. 36, no. 2, pp. 1–20, Apr. 2019.

[39] F. Sadeghi and S. Levine, "CAD2RL: Real single-image flight without a single real image," 2016, *arXiv:1611.04201*.

[40] 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 *Proc. Int. Conf. Robot. Automat. (ICRA)*, May 2019, pp. 6008–6014.

[41] H. Hu, K. Zhang, A. H. Tan, M. Ruan, C. G. Agia, and G. Nejat, "A sim-to-real pipeline for deep reinforcement learning for autonomous robot navigation in cluttered rough terrain," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 6569–6576, Oct. 2021.

[42] Y. Zhu, Z. Wang, C. Chen, and D. Dong, "Rule-based reinforcement learning for efficient robot navigation with space reduction," *IEEE/ASME Trans. Mechatronics*, early access, Apr. 13, 2021, doi: 10.1109/TMECH.2021.3072675.

[43] A. A. Rusu, M. Vecerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *Proc. 1st Annu. Conf. Robot Learn. (CoRL)*, Mountain View, CA, USA, Nov. 2017, pp. 262–270. [Online]. Available: https://proceedings.mlr.press/v78/rusu17a.html

[44] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2018, pp. 1–8.

[45] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *Proc. 2nd Annu. Conf. Robot Learn. (CoRL)*, Zürich, Switzerland, Oct. 2018, pp. 734–743. [Online]. Available: https://proceedings.mlr.press/v87/matas18a.html

[46] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 12627–12637.

[47] W. Yuan, K. Hang, D. Kragic, M. Y. Wang, and J. A. Stork, "End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer," *Robot. Auto. Syst.*, vol. 119, pp. 119–134, Sep. 2019.

[48] R. Liu, F. Nageotte, P. Zanne, M. de Mathelin, and B. Dresp-Langley, "Deep reinforcement learning for the control of robotic manipulation: A focussed mini-review," *Robotics*, vol. 10, no. 1, p. 22, Jan. 2021.

[49] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," 2016, *arXiv:1610.03518*.

[50] A. Mandlekar, Y. Zhu, A. Garg, L. Fei-Fei, and S. Savarese, "Adversarially robust policy learning: Active construction of physically-plausible perturbations," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 3932–3939.

[51] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 2817–2826.

[52] M. Wulfmeier, I. Posner, and P. Abbeel, "Mutual alignment transfer learning," in *Proc. 1st Annu. Conf. Robot Learn.*, vol. 78, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., Nov. 2017, pp. 281–290. [Online]. Available: https://proceedings.mlr.press/v78/wulfmeier17a.html

[53] F. Muratore, F. Treede, M. Gienger, and J. Peters, "Domain randomization for simulation-based policy optimization with transferability assessment," in *Proc. Conf. Robot Learn.*, 2018, pp. 700–713.

[54] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *Proceedings of Robotics: Science and Systems*. Pittsburgh, PA, USA: RSS Foundation, Jun. 2018, pp. 10–20.

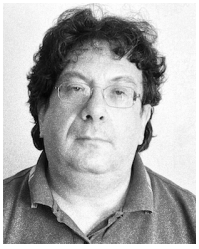
[55] F. Golemo, A. A. Taiga, A. Courville, and P.-Y. Oudeyer, "Sim-to-real transfer with neural-augmented robot simulation," in *Proc. Conf. Robot Learn.*, 2018, pp. 817–828.

[56] J. Van Baar, A. Sullivan, R. Cordorel, D. Jha, D. Romeres, and D. Nikovski, "Sim-to-real transfer learning using robustified controllers in robotic tasks involving complex dynamics," in *Proc. Int. Conf. Robot. Automat. (ICRA)*, May 2019, pp. 6001–6007.

[57] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving Rubik's cube with a robot hand," 2019, *arXiv:1910.07113*.
[58] X. Pan, D. Seita, Y. Gao, and J. Canny, "Risk averse robust adversarial reinforcement learning," in *Proc. Int. Conf. Robot. Automat. (ICRA)*, May 2019, pp. 8522–8528.
[59] K. G. S. Apuroop, A. V. Le, M. R. Elara, and B. J. Sheu, "Reinforcement learning-based complete area coverage path planning for a modified hTrihex robot," *Sensors*, vol. 21, no. 4, p. 1067, Feb. 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/4/1067
[60] J. P. Hanna and P. Stone, "Grounded action transformation for robot learning in simulation," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 1–7.
[61] J. Siekmann, K. Green, J. Warila, A. Fern, and J. Hurst, "Blind bipedal stair traversal via Sim-to-Real reinforcement learning," 2021, *arXiv:2105.08328*.
[62] 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," 2021, *arXiv:2103.14295*.
[63] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: An overview," in *Proc. SAI Intell. Syst. Conf.* Cham, Switzerland: Springer, 2016, pp. 426–440.
[64] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2012, pp. 5026–5033.
[65] Blender Community. (2020). *Blender: Open Source 3D Modeling Suit*. [Online]. Available: https://www.blender.org
[66] Sourceforge. (2020). *SimSpark*. [Online]. Available: https://simspark.sourceforge.net/
[67] Open Source Robotics Foundation. (2020). *GAZEBO: Robot Simulation Made Easy*. [Online]. Available: https://www.gazebosim.org/
[68] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "DART: Dynamic animation and robotics toolkit," *J. Open Source Softw.*, vol. 3, no. 22, p. 500, Feb. 2018.
[69] CM Labs. (2001). *Vortex Studio Real-Time Simulation and Visualization Software for System-Level Modeling of Mechatronics and Mechanical Equipment*. [Online]. Available: https://www.cm-labs.com/vortex-studio/
[70] E. Coumans and Y. Bai. (2016–2019). *Bullet Physics SDK*. [Online]. Available: https://bulletphysics.org
[71] E. Coumans and Y. Bai. (2016). *Pybullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning*. [Online]. Available: https://pybullet.org
[72] F. Xia, A. Zamir, Z. He, S. Sax, J. Malik, and S. Savarese, "Gibson ENV: Real-world perception for embodied agents," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9068–9079.
[73] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner. (2000). *TORCS, the Open Racing Car Simulator*. [Online]. Available: https://torcs.sourceforge
[74] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," 2016, *arXiv:1606.04671*.
[75] 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 *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2017, pp. 23–30.
[76] S. James, A. J. Davison, and E. Johns, "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task," in *Proc. 1st Annu. Conf. Robot Learn. (CoRL)*, Mountain View, CA, USA, Nov. 2017, pp. 334–343. [Online]. Available: https://proceedings.mlr.press/v78/james17a.html
[77] I. Mordatch, K. Lowrey, and E. Todorov, "Ensemble-CIO: Full-body dynamic motion planning that transfers to physical humanoids," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2015, pp. 5307–5314.
[78] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Optimizing walking controllers for uncertain inputs and environments," *ACM Trans. Graph.*, vol. 29, no. 4, p. 73, 2010.
[79] A. Vandesompele, G. Urbain, H. Mahmud, F. Wyffels, and J. Dambre, "Body randomization reduces the sim-to-real gap for compliant quadruped locomotion," *Frontiers Neurorobotics*, vol. 13, p. 9, Mar. 2019, doi: 10.3389/fnbot.2019.00009.
[80] S. P. Bhattacharyya, "Robust control under parametric uncertainty: An overview and recent results," *Annu. Rev. Control*, vol. 44, pp. 45–77, Jan. 2017.
[81] J. Ackermann, *Robust Control: The Parameter Space Approach*. London, U.K.: Springer, 2012.
[82] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," 2016, *arXiv:1606.01540*.
[83] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.
[84] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5048–5058.
[85] N. Heess, J. J Hunt, T. P Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," 2015, *arXiv:1512.04455*.
[86] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[87] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[88] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," 2017, *arXiv:1707.08817*.
[89] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Proc. The 2nd Conf. Robot Learn.*, vol. 87, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., Oct. 2018, pp. 651–673. [Online]. Available: https://proceedings.mlr.press/v87/kalashnikov18a.html
[90] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[91] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 41–48.
[92] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, San Diego, CA, USA, May 2015, pp. 1–11.
[93] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proc. 2nd Int. Conf. Learn. Represent. (ICLR)*, Banff, AB, Canada, Apr. 2014, pp. 1–10.
[94] B. F. Hobbs and A. Hepenstal, "Is optimization optimistically biased?" *Water Resour. Res.*, vol. 25, no. 2, pp. 152–160, Feb. 1989.
[95] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Mach. Learn.*, vol. 13, no. 1, pp. 103–130, 1993.
[96] S. P. Choi and D.-Y. Yeung, "Predictive Q-routing: A memory-based reinforcement learning approach to adaptive traffic control," in *Proc. Adv. Neural Inf. Process. Syst.*, 1996, pp. 945–951.
[97] J. Peng and R. J. Williams, "Efficient learning and planning within the Dyna framework," *Adapt. Behav.*, vol. 1, no. 4, pp. 437–454, 1993.
[98] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 1125–1134.
[99] W. Uther and M. Veloso, "Adversarial reinforcement learning," Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-03-107, 1997.
[100] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine Learning Proceedings*. Amsterdam, The Netherlands: Elsevier, 1994, pp. 157–163.
[101] D. L. Ma, S. H. Chung, and R. D. Braatz, "Worst-case performance analysis of optimal batch control trajectories," in *Proc. Eur. Control Conf. (ECC)*, Aug. 1999, pp. 3256–3261.
[102] D. L. Ma and R. D. Braatz, "Worst-case analysis of finite-time control policies," *IEEE Trans. Control Syst. Technol.*, vol. 9, no. 5, pp. 766–774, Sep. 2001.

[103] J. C. Doyle, K. Glover, P. P. Khargonekar, and B. A. Francis, "State-space solutions to standard H_2 and H_∞ control problems," *IEEE Trans. Autom. Control*, vol. 34, no. 8, pp. 831–847, Aug. 1989.
[104] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4026–4034.
[105] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learn. Res.*, vol. 10, pp. 1633–1685, Jul. 2009.
[106] N. Hansen, "The CMA evolution strategy: A comparing review," in *Towards a New Evolutionary Computation*. Berlin, Germany: Springer, 2006, pp. 75–102.
[107] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, pp. 1–7.
[108] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2018, pp. 1–8.
[109] A. Farchy, S. Barrett, P. MacAlpine, and P. Stone, "Humanoid robots learning to walk faster: From the real world to simulation and back," in *Proc. Int. Conf. Auto. Agents Multi-Agent Syst.*, 2013, pp. 39–46.
[110] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.
[111] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1329–1338.
[112] 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 *Proc. IEEE Int. Conf. Rehabil. Robot.*, Jun. 2011, pp. 1–5.
[113] Poppy Station. (2020). *Poppy Project: Open Source Robotics Platform*. [Online]. Available: https://www.poppy-project.org/en/
[114] F. Muratore, C. Eilers, M. Gienger, and J. Peters, "Data-efficient domain randomization with Bayesian optimization," 2020, *arXiv:2003.02471*.
[115] N. Schweighofer and K. Doya, "Meta-learning in reinforcement learning," *Neural Netw.*, vol. 16, no. 1, pp. 5–9, 2003.
[116] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 5302–5311.
[117] L. Kirsch, S. van Steenkiste, and J. Schmidhuber, "Improving generalization in meta reinforcement learning using learned objectives," 2019, *arXiv:1910.04098*.
[118] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proc. 28th Int. Conf. Mach. Learn. (ICML)*, 2011, pp. 465–472.
[119] M. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 2, pp. 408–423, Feb. 2013.
[120] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," 2020, *arXiv:2004.05439*.
[121] S. Tu and B. Recht, "Least-squares temporal difference learning for the linear quadratic regulator," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 5005–5014.

ERICA SALVATO received the B.Sc. degree in electronic engineering from the University of Messina, Italy, in 2015, and the M.Sc. degree in electrical and control systems engineering from the University of Trieste, Italy, in 2018. She is currently pursuing the Ph.D. degree with the Department of Engineering and Architecture, University of Trieste. Her research interests include the artificial intelligence application as a systems control tool but includes also control theory, machine learning, and robotics.



GIANFRANCO FENU received the Laurea degree (M.Sc. degree) in electronic engineering and the Ph.D. degree in information engineering from the University of Trieste, in 1996 and 2001, respectively. From October 1997 to March 1998, he was a Visiting Ph.D. Student with the Fachgebiet Mess- und Regelungstechnik (Institute for Measurement and Automatic Control), University of Duisburg–Essen, Germany, led by Prof. Paul M. Frank. Since November 1999, he has been an Assistant Professor with the Department of Engineering and Architecture, University of Trieste. His research interests include control theory, fault diagnosis, machine learning, and robotics.

ERIC MEDVET received the degree *(cum laude)* in electronic engineering and the Ph.D. degree in computer engineering from the University of Trieste, Italy, in 2004 and 2008, respectively. He is currently with the University of Trieste as an Associate Professor in computer engineering and the Director of the Evolutionary Robotics and Artificial Life Laboratory. Besides evolutionary robotics, his research interests include genetic programming and applications of machine learning.

FELICE ANDREA PELLEGRINO (Member, IEEE) was born in Conegliano, Italy, in 1974. He received the Laurea degree in engineering and the Ph.D. degree from the University of Udine, Italy, in June 2000 and May 2005, respectively. Since 2006, he has been with the Department of Engineering and Architecture, University of Trieste, where he is currently an Associate Professor. His research interests include control theory, machine learning, and computer vision. He is also an Associate Editor of the IEEE Control Systems Letters and a member of the Conference Editorial Board of the IEEE Control Systems Society and the European Control Association.