García, J. and Fernández, F. Probabilistic Policy Reuse for Safe Reinforcement Learning. ACM Transactions on Autonomous and Adaptive Systems, 13(3), (2019)

DOI: https://doi.org/10.1039/c3lc51419f

Probabilistic Policy Reuse for Safe Reinforcement Learning

JAVIER GARCÍA* and FERNANDO FERNÁNDEZ*, Universidad Carlos III de Madrid, Spain

This work introduces *Policy Reuse for Safe Reinforcement Learning* (PR-SRL), an algorithm that combines Probabilistic Policy Reuse and teacher advice for safe exploration in dangerous and continuous state and action reinforcement learning problems in which the dynamic behavior is reasonably smooth, and the space is Euclidean. The algorithm uses a continuously increasing monotonic risk function which allows for the identification of the probability to end up in failure from a given state. Such a risk function is defined in terms of how far such a state is from the state space known by the learning agent. Probabilistic Policy Reuse is used to safely balance the exploitation of actual learned knowledge, the exploration of new actions, and the request of teacher advice in considered dangerous parts of the state space. Specifically, the π-reuse exploration strategy is used. Using experiments in the helicopter hover task and a business management problem, we show that the π-reuse exploration strategy can be used to completely avoid the visit to undesirable situations, while maintaining the performance (in terms of the classical long-term accumulated reward) of the final policy achieved.

CCS Concepts: • **Theory of computation → Reinforcement learning**; • **Computing methodologies →** *Temporal difference learning*; Control methods.

Additional Key Words and Phrases: Reinforcement Learning, Case-based Reasoning, Software Agents

ACM Reference Format:
Javier García and Fernando Fernández. 2018. Probabilistic Policy Reuse for Safe Reinforcement Learning. 1, 1 (May 2018), 24 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

While most Reinforcement Learning (RL) tasks [28] are focused on maximizing a long-term cumulative reward, RL researchers are also paying increasing attention to the safety of the approaches (e.g., avoiding visits to undesirable situations, collisions, crashes, etc.) during the training process [9, 10]. Thus, when using RL techniques in dangerous control tasks, an important question arises; namely, how can we ensure that the exploration of the state-action space will not cause damage or injury while, at the same time, learning (near-)optimal policies? The matter, in other words, is one of ensuring that the agent is able to explore a dangerous environment both safely and efficiently.

This safe exploration of the search space is particularly interesting in tasks such as robot control or navigation. In these tasks, practical deployment of learning algorithms must contend with the fact that the training process itself may be unsafe for the robot. For instance, in the self-driving task [17, 20], in which an agent must learn to autonomously drive a vehicle (e.g., a car or helicopter), the agent is likely to continuously crash until it learns how to do it. However, it would be highly desirable to avoid the visit to undesirable situations during the whole training process. One could

*Both authors contributed equally to this research.

Authors' address: Javier García, fjgpolo@inf.uc3m.es; Fernando Fernández, ffernand@inf.uc3m.es, Universidad Carlos III de Madrid, Avenida de la Universidad, 30, Leganés, Spain, 28911.

XXXX-XXXX/2018/5-ART $15.00
https://doi.org/10.1145/1122445.1122456

argue that such a training process may be performed first in simulation (where it is not necessary to behave safely) and, afterwards, apply the learned safe policy to the real environment. This presents two drawbacks. On one hand, such simulators are not always available. On the other hand, behaviors learned in simulation are not always transferable to real environments due to a significant drop of their performance [22]. Simulation cannot reproduce real conditions accurately accurately enough, and, hence, a safe policy trained with simulation could have catastrophic consequences tested in the real environment. In this case, the agent should keep learning continuously and adaptively in the real environment to overcome the imperfections of the simulation. As well, such a training process should be conducted safely. In fact, RL is inherently an online learning setting where the agent does not stop learning, so generally there is no testing phase [28].

There are many RL-based approaches to learn robotic tasks, but most of them do not face the problem of avoiding undesirable situations during training [4, 13, 17, 29]. Commonly in these approaches, the agent receives a large negative reward for each visit to an undesirable or dangerous situation, and it must repeatedly experience these (possibly catastrophic) situations to learn how to avoid them. Moreover, in these approaches, it is usually used exploration/exploitation strategies such as ϵ-greedy that may even result in constant visits to these situations (especially where there is a high probability of random action selection). However, the visits to dangerous states are unfeasible in most real domains, and, consequently, such visits should be avoided from the earliest steps of the training process.

Policy search [5] and evolutionary approaches [16, 20] have also been used to successfully learn robot control tasks. In contrast to the previous approaches, the aim of these is to search directly across policies rather than approximating a value function. They use a global value (a "fitness") which measures the ability of the robot to perform a given behavior. In this context, Levine et al. [18] propose a policy search method to learn policies that map raw image observations directly to torques at the robot's motors. In this case, a stochastic gradient descent method is used to train a deep neural network representing the policy. Instead, evolutionary approaches [16, 20, 21] are usually based on evolving populations of neural networks (each one representing a different policy) with the aim of finding a better policy in each generation. However, the random generation of the initial population and the random mutation of the neural networks during evolution are made to visit undesirable states repeatedly [9]. Therefore, most of these approaches suffer from the same problem: the absence of mechanisms for avoiding the visit to undesirable situations during the exploration of the state and action spaces.

One way to explore the state and action spaces in a safe manner is by using prior knowledge about the task: some kind of prior knowledge is clearly beneficial, e.g., if particular actions lead to dangerous situations, we would like to know a priori what these actions are, so that we do not select them during training. One way to provide such prior knowledge is by the demonstrations provided by a demonstrator or teacher, who is assumed to have a safe behavior. All approaches falling under this category are framed according to the field of *Learning from Demonstration* (LfD) [2]. In most of these approaches the primary objective is concerned with matching the performance of the teacher [12, 25]. In this case, the safe exploration is achieved by the teacher who says at every moment what action should be executed, so that dangerous situations are avoided. In other cases, such demonstrations are used to bootstrap the learning algorithm (i.e., as an initialization procedure), before using traditional learning techniques, such as exploration-based methods like RL [15, 24]. However, such subsequent exploration processes are conducted in an unsafe way. Therefore, the main drawbacks of these approaches are: (i) learner performance is heavily limited by the quality of the teacher's demonstrations, and (ii) in all these works no explicit definition of risk is ever given. In this paper, although a teacher is also used, the objective is to explore beyond what is provided in the teacher demonstrations in an efficient and safe manner. Few research

studies have been conducted on the safe exploration of the state and action space given an explicit definition of the risk concept and how we should handle it.

One of these algorithms is the ***P**olicy **I**mprovement through **S**afe **R**einforcement **L**earning* (PI-SRL) algorithm [9]. PI-SRL is an algorithm for safe exploration in dangerous and continuous control tasks. Such a method requires a predefined (and safe) baseline policy, provided by a teacher, which is assumed to be suboptimal (otherwise, learning would be pointless). The method has shown the best performance in all the domains evaluated when compared with previous approaches, like the evolutionary RL approach selected winner of the helicopter domain in the 2009 RL Competition [20] and Geibel and Wysotzki's risk-sensitive RL approach [10]. PI-SRL is based on a risk function, $\varrho^B(s)$, that measures the risk of a state in terms of its similarity to previously visited states in a case base, B. In particular, the risk function divides the state space into known and unknown states. A state is considered a known state if it is similar (or equal) to a state previously visited, and it is unknown otherwise. In other words, a state is considered known if the distance to the closest state in B is lower than the value of a parameter θ, and unknown otherwise. Moreover, we consider that an unknown state is an unsafe state because the agent does not know what action to perform on it. Certainly, not knowing what to do in a given situation increases the probability of ending up in trouble. Thus, the concept of *risk* is associated to the concept of *unknown* [9]. In case of an unknown state, the risk is maximum, and the agent has to ask the teacher for advice. Then, this unknown state together with the action suggested by the teacher is stored in B, and the unknown state becomes a known state. In this way, if the agent revisits a state similar (or equal) to the new known state, it will know what action to perform. Therefore, in this case, the risk function is defined as a binary step function: the risk is minimal for a known state, and it is the maximum for an unknown state. However, such binary step risk function may still produce damages in the learning agent. The reason is that to follow the teacher advice only when the risk is the maximum may be too late to avoid dangerous situations. Although PI-SRL reduces the number of visits to undesirable situations compared with other exploration strategies [9], it is not able to completely avoid these visits in domains in which it is strictly necessary. For example, consider an agent learning to pilot a real helicopter. We cannot afford the agent to crash the helicopter during learning, because after the first impact we probably will not have a helicopter for further learning.

In contrast to the binary step risk function used by PI-SRL, one would expect that the risk function is a continuously increasing monotonic function. In such a way, while the limit of θ is approaching, the risk should start to grow, and the learning agent could start to use the teacher advice. For instance, returning to the self-driving helicopter, let's assume that an unknown state is reached where the helicopter is about to crash. In this situation, the teacher is asked for advice, but it is too late to avoid the accident. It only postpones the inevitable. Therefore, in this case, it would be advisable to increase the probability of asking the teacher for advice as we approach these unknown and potentially dangerous situations, rather than wait until the last moment, when there is no possibility of avoiding the catastrophe. For this reason, we propose the use of a continuously increasing monotonic risk function that determines the probability to follow the teacher advice instead of a binary step one. To integrate such advice, Probabilistic Policy Reuse is used, specifically, the π-reuse exploitation strategy. In its initial definition, the π-reuse strategy is an exploration strategy able to bias a new learning process with a previously acquired policy [6, 7], but recent works have also suggested its use to incorporate human demonstrations [31], or even teacher advice [32]. Taylor et al. [31] define different ways to reuse the policy learned from the teacher, one of which is the π-reuse exploration strategy. The problem is that this reuse strategy does not take into account the confidence on the part of teacher (or the learner) to decide when to incorporate a teacher advice. Torrey and Taylor [32] use confidence measures to compute state-specific advice probabilities. In that work, in states where the teacher has much higher confidence than the learner,

it gives advice with a higher probability. In contrast to these approaches, we use the reuse strategy in a different way. In our case, the confidence of the learner in a specific state is given by the probability of that state to be considered as a *known* state. Otherwise, a teacher advice is required. Therefore, in our work, we do not consider the confidence of the teacher: we assume the teacher provides safe actions, or, at the very least, safer than ones obtained through random exploration. We use the reuse strategy to incorporate such advice in a safe exploration process.

Additionally, in this paper, we have improved the learning process of the *PI-SRL* algorithm [9], which was composed of two main steps: *Modeling Baseline Behaviors by Case-Based Reasoning (CBR)* [1], where a case base policy was learned using experiences extracted from teacher's executions; and *Improving the Learned Base Line Behavior*, where previously acquired case based policy was improved. Instead, the new algorithm proposed, Policy Reuse for Safe Reinforcement Learning (*PR-SRL*) demonstrates that the first step can be skipped by assuming that the initial case base policy to improve is empty. It will produce that, at the beginning of the learning, only teacher advices will be executed. But progressively, while the case base grows, teacher advice will decrease.

Therefore, this paper proposes three new contributions: (i) the use of a continuously increasing monotonic risk function, (ii) the integration of the risk function within the π-reuse strategy for safe exploration of the state and action spaces, and (iii) the unification of the two steps of the PI-SRL algorithm in only one step. We expect that these contributions allow us to completely avoid the visit of undesirable situations in domains in which it is strictly necessary. Particularly, we demonstrate empirically the strength of the new contributions in two simulated domains: the helicopter hover task from the RL Competition [23], and a business simulator [3]. In these domains, we propose to learn a near-optimal policy avoiding the helicopter crashes and company bankrupts during the learning phase.

Next, the main concepts of Safe Reinforcement Learning are described. This section reprises the descriptions given in [9] (Section 2) for ease of reference. Then, Section 3 describes the proposed continuous risk function, and Section 4 describes the π-reuse exploration strategy. Afterwards, Section 5 describes the algorithms proposed. Finally, Section 6 reports the evaluation performed, and Section 7 summarizes the main conclusions of this manuscript and discusses the limitations and domain applicability restrictions of the proposed approach.

2 SAFE REINFORCEMENT LEARNING

To illustrate the concept of safety used in our approach, a navigation problem is presented in Figure 1. In the navigation problem presented in Figure 1, a control policy must be learned to get from a particular start state to a goal state, given a set of demonstration trajectories. In this environment, we assume the task to be difficult due to a stochastic and complex dynamic of the environment (e.g., an extremely irregular surface in the case of a robot navigation domain or wind effects in the case of the helicopter hover task). This stochasticity makes it impossible to complete the task using exactly the same trajectory every time. Additionally, the problem supposes that a set of demonstrations from a baseline controller performing the task (the continuous black lines) are also given. This set of demonstrations is composed of different trajectories covering a well-defined region of the state space (the region within the rectangle), where each of these trajectories is the succession of the states visited by performing the policy of the baseline behavior. Note that the policy of this baseline behavior is deterministic: given the same situation s, the agent will always perform the same action a. However, given the stochasticity of the environment (e.g., irregular surface, wind), the performance of the same action a in a particular state s does not always lead the agent to the same situation, hence, each execution of this deterministic policy leads the agent through a different trajectory. In any case, it is also necessary to add an additive exploration noise

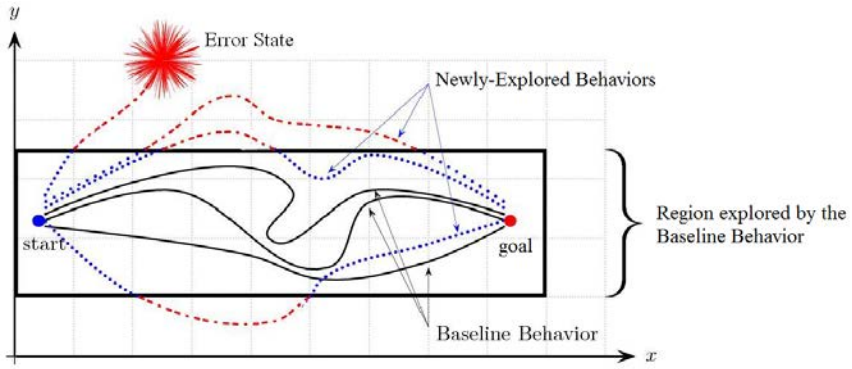

Fig. 1. Exploration strategy based on adding small amounts of noise to a baseline policy behavior. The baseline behavior is shown by solid lines and the new explored behaviors are shown by dotted lines.

to the actions provided by the baseline behavior in order to try different actions in the same state. Thus, we can choose the best of them.

Our approach is based on the addition of small amounts of Gaussian noise or perturbations to the baseline trajectories (i.e., to the actions provided by the baseline behavior) in order to find new and better ways of completing the task. This noise will affect the baseline trajectories in different ways, depending on the amount of noise added, which, in turn, depends on the amount of risk to be taken. If no risk is desired, the noise added to the baseline trajectories will be 0, and, consequently, no new or improved behavior will be discovered (nevertheless, the robot will never fall off the cliff and the helicopter will never crash). If, however, an intermediate level of risk is desired, small amounts of noise will be added to the baseline trajectories, and new trajectories (the dotted blue lines) to complete the task are discovered. In some cases, the exploration of new trajectories leads the robot to unknown regions of the state space (the dashed red lines). The robot is assumed to be able to detect such situations with a risk function and use the baseline behavior to return to safe, known states. If, instead, a very high risk is desired, large amounts of noise will be added to the baseline trajectories, leading to the discovery of new trajectories (but also to a higher probability that the robot gets damaged). The iteration of this process leads the robot to progressively and safely explore the state and action spaces in order to find new and improved ways to complete the task. The degree of safety in the exploration, however, will depend on the risk taken.

2.1 Error and Non-Error States

In this paper, we follow as far we can the notation presented in Geibel et al. [10] for the definition of our concept of *risk*. In their study, Geibel et al. [10] associate risk with *error states* and *non-error states*, with the former understood as a state in which it is considered undesirable or dangerous to enter.

Definition 2.1. **(Error and Non-Error States)**. Let S be a set of states and $\Phi \subset S$ the set of error states. A state $s \in \Phi$ is an undesirable terminal state where the control of the agent ends when s is reached with damage or injury to the agent, the learning system or any external entities. The set $\Gamma \subset S$ is considered a set of non-error terminal states with $\Gamma \cap \Phi = \emptyset$ and where the control of the agent ends normally without damage or injury.

In terms of RL, if the agent enters an error state, the current episode ends with damage to the learning system (or other systems); whereas if it enters a non-error state, the episode ends normally

and without damage. Thus, Geibel et al. define the risk of s with respect to policy π, $\rho^{\pi}(s)$, as the probability that the state sequence $(s_i)_{i\geq 0}$ with $s_0 = s$, generated by the execution of policy π, terminates in an error state $s' \in \Phi$. By definition, $\rho^{\pi}(s) = 1$ if $s \in \Phi$. If $s \in \Gamma$, then $\rho^{\pi}(s) = 0$ because $\Phi \cap \Gamma = \emptyset$. For states $s \notin \Phi \cup \Gamma$, the risk taken depends on the actions selected by the policy π. With these definitions, we have the theoretical framework with which to introduce our own definition of the risk associated with *known* and *unknown states*.

2.2 Known and Unknown States in Continuous Action and State Spaces

We assume a continuous, n-dimensional state space $S \subset \Re^n$, where each state $s = (s_1, s_2, \ldots, s_n) \in S$ is a vector of real numbers, and each dimension has its individual domain $D_i^s \subset \Re$. In a similar way, we assume a continuous and m-dimensional action space $A \subset \Re^m$, where each action $a = (a_1, a_2, \ldots, a_m) \in A$ is a vector of real numbers, and each dimension has its individual domain $D_i^a \subset \Re$. Additionally, the agent we consider here is endowed with a memory or case-base $B = \{c_1 \ldots, c_\eta\}$ of size η. Each memory element c_i represents some cases the agent has experienced before. Such cases, as is typical in CBR [1], consist of a problem part (i.e., a description of a problem situation) and a solution part (i.e., how one has reacted or solved that problem). In our case, the problem part corresponds to a state, and the solution part to the action performed in that state.

Definition 2.2. **(Case Base)**. A case-base is a set of cases $B = \{c_1 \ldots, c_\eta\}$. Every case c_i consists of a state-action pair (s_i, a_i) the agent has experienced in the past and with an associated value $V(s_i)$. Thus, $c_i =< s_i, a_i, V(s_i) >$, where the first element represents the case's problem part and corresponds to the state s_i, the following element a_i depicts the case solution (i.e., the action expected when the agent is in the state s_i) and the final element $V(s_i)$ is the value function associated with the state s_i.

Hence, the cases in B describe a **Case Based Policy** of the agent, π_B^{θ}, and its associated value function $V^{\pi_B^{\theta}}$. When the agent receives a new state s_q, the agent first retrieves the nearest neighbor to the s_q point in B according to some similarity metric and then the associated action is performed. In this paper we consider the Euclidian distance as similarity metric (Equation 1).

$$d(s_q, s_i) = \sqrt{\sum_{j=0}^{n} (s_{q,j} - s_{i,j})^2} \tag{1}$$

The Euclidean distance metric is useful when dynamic behavior of the system is reasonably smooth [26]. However, many researchers have begun to ask how the distance metric itself can learn or adapt in order to achieve better results in domains where such assumption may not be true [30]. While the use of distance metric learning techniques would certainly be desirable in order to induce a more powerful distance metric for a specific domain, such a consideration lies outside the scope of the present study. In this paper, we have focused only on Euclidean tasks in which Euclidean distance has been previously proven successful [8, 19].

Traditionally, case base approximations use a *density threshold*, θ, which is used to determine when a new case should be added to the memory. When the distance of the nearest neighbor to s_q is greater than θ, a new case is added to the memory. In this sense, the parameter θ defines the size of the classification region for each case in B.

Definition 2 .3. **(Known and Unknown States)** Given a case base $B = \{c_1 \ldots, c_\eta\}$ composed of cases $c_i = (s_i, a_i, V(s_i))$, a state s_q is considered **known**, when $\min_{1\leq i\leq \eta} d(s_q, s_i) \leq \theta$; the state s_q is considered **unknown** otherwise. Formally, we consider a set $\Omega \subseteq S$ of **known** states, and we allow an additional set of **unknown** states $\Upsilon \subseteq S$ with $\Omega \cap \Upsilon = \emptyset$, and $\Omega \cup \Upsilon = S$.

Therefore, from Definition 2.3. a state s_q is considered known if a case is available for it in the case base, B, and unknown otherwise. Thus, with Definition 2.3, states can be identified as known or unknown. When the agent receives a new state $s \in \Omega$, it performs the action a_i of the case c_i for which $d(s, s_i) = \min_{1 \leq j \leq \eta} d(s, s_j)$. However, if the agent receives a state $s \in \Upsilon$ where, by definition, the distance to any state in B is larger than θ, no case is retrieved. Consequently, the action to be performed from that state is unknown to the agent. It is important to bear in mind that at the beginning of the exploration process, the agent does not have any information about the environment, hence, all the states are unknown, i.e., at the beginning of the exploration process it is $\Upsilon = S$.

2.3 Supporting the Exploration Process

As this paper supposes that a teacher is available for the task to be learned, the teacher is taken as the baseline behavior. Although some studies have examined the use of robotic teachers, hand-written control policies and simulated planners, the great majority to date have made use of human teachers. However, in our case and depending on the task, the teacher may be queried for hours, with queries every few seconds. For this reason, in this paper we use suboptimal automatic controllers as teachers (i.e., we are not using human teachers, but pre-built software agents that can be considered as a black box that when given a state returns a suboptimal action to be performed in that state), with π_T taken as the teacher's policy.

Definition 2.4. **(Baseline behavior)**. Policy π_T is considered the baseline behavior about which three assumptions are made: (i) it is able to provide safe demonstrations of the task to be learnt; (ii) it is able to support the subsequent exploration process, advising suboptimal actions in unknown states to reduce the probability of entering into error states and return the system to a known situation; and (iii) its performance is below the optimal.

While optimal baseline behaviors are certainly ideal to behave safely, non-optimal behaviors are often easy (or easier) to implement or generate than optimal ones. In this work, π_T is used to support the exploration process conducted to improve the abilities of the baseline behavior. As the exploration process continues, an action of π_T is requested only when required, that is, when the agent is in an unknown state. In this step, π_T acts as a backup policy in the case of an unknown state with the intention of guiding the learning away from catastrophic errors or, at least, reducing their frequency. It is important to note that the baseline behavior cannot demonstrate the correct action for every possible state. However, while the baseline behavior might not be able to indicate the best action in all cases, the action it supplies should, at the very least, be safer than that obtained through random exploration. In this way, we avoid the random exploration of actions in unknown states that might be potentially dangerous.

2.4 The Risk Parameter

In order to maximize exploration safety, it seems advisable that movement through the state space not be arbitrary, but rather that known space be expanded only gradually by starting from a known state. Such an exploration is carried out through the perturbation of the state-action trajectories generated by the policy π_B^θ. Perturbation of the trajectories is accomplished by the addition of Gaussian random noise to the actions in B in order to obtain new ways of completing the task. Thus, the Gaussian exploration takes place around the current approximation of the action a_i for the current known state $s_c \in \Omega$, with $c_i = (s_i, a_i, V(s_i))$ and $d(s_c, s_i) = \min_{1 \leq j \leq \eta} d(s, s_j)$. The action performed is sampled from a Gaussian distribution with the mean at the action output given by the instance selected in B. When a_i denotes the algorithm action output, the probability of selecting action a'_i, $\pi(s, a'_i)$ is computed using Equation 2.

$$\pi(s, a_i') = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(a_i'-a_i)^2/2\sigma^2} \quad \text{if } \sigma^2 > 0. \tag{2}$$

The shape of the Gaussian distribution depends on parameter σ (standard deviation). In this study, σ is used as a *width parameter*. While large σ values imply a wide bell-shaped distribution, increasing the probability of selecting actions a_i' very different from the current action a_i, a small σ value implies a narrow bell-shaped distribution, increasing the probability of selecting actions a_i' very similar to the current action a_i. When $\sigma^2 = 0$, we assume $\pi(s, a_i) = 1$. Hence, the σ value is directly related to the amount of perturbation added to the state-action trajectories generated by the policy π_B^θ. Higher σ values imply greater perturbations (more Gaussian noise) and a greater probability of visiting unknown states.

This form of exploration promotes only exploration in the neighborhood of the baseline policy, π_B^θ, hence, it removes the need for global exploration of the entire state-space of the RL problem. It allows the algorithm to converge faster on the (near-)optimal policy while reducing the number of visits to undesirable states, but it also can get stuck in local optima. Large σ values may be used to avoid being trapped in such local optima, but it may involve visiting more undesirable situations. Therefore, the user can gradually increase from run to run the value of the risk parameter σ in order to obtain better policies as described in Section 6.1, whilst also assuming a greater likelihood of damage in the learning system. Nevertheless, it is important to note that any ultimate decision about which level of risk is assumed depends on the criteria of the researcher. If, for instance, the minimization of the number of failures is deemed the most important optimization criterion, a small σ value should be selected. Similarly, if the maximization of the cumulative reward is instead judged to be the most important optimization criterion (independently of the number of failures generated), a large σ value should be selected to increase the likelihood the policy is closer to the global optima.

3 THE CONTINUOUS RISK FUNCTION

A main contribution of this work is the definition of a continuous risk function. To highlight the difference from the discrete one, defined in a previous work [9], it is described next.

Definition 3.1. **(Discrete Case Based Risk Function)** Given a case base $B = \{c_1 \ldots, c_\eta\}$ composed of cases $c_i = (s_i, a_i, V(s_i))$, the risk for each state s is defined as Equation 3.

$$\varrho^{\pi_B^\theta}(s) = \begin{cases} 0 & \text{if } \min_{1 \le j \le \eta} d(s, s_j) < \theta \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

Thus, $\varrho^{\pi_B^\theta}(s) = 1$ holds if $s \in \Upsilon$. If $s \in \Omega$, then $\varrho^{\pi_B^\theta}(s) = 0$. When the risk is high, the policy of the teacher should be followed to avoid damages in the learning system. A continuous version of the risk function is defined in Equation 4, while Figure 2 plots both functions in terms of the $\min_{1 \le j \le \eta} d(s, s_j)$ factor, for $\theta = 0.3$ and $k = 6$.

Definition 3.2. **(Continuous Case Based Risk Function)** Given a case base $B = \{c_1 \ldots, c_\eta\}$ composed of cases $c_i = (s_i, a_i, V(s_i))$, the risk for each state s is defined as Equation 4.

$$\varrho^B(s) = 1 - \frac{1}{1 + e^{\frac{k}{\theta}((\min_{1 \le j \le \eta} d(s, s_j) - \frac{\theta}{k}) - \theta)}} \tag{4}$$

Equation 4 allows us to obtain a smoother transition between risk-free states (i.e., known states) and risk states (i.e., unknown states). The parameter k has a double effect. On one hand, depending on its value, the width of the sigmoid function varies. The lower the value of k is, the wider the sigmoid function will be. This implies a wider yellow region on the left of the parameter θ in Figure 2, i.e., a higher probability to consider known states as unknown states. This results in a less

aggressive exploration of the state space during the learning process since the teacher advice are more frequently required (being able to affect negatively the final performance of the algorithm). This also implies a wider orange region on the right of the parameter θ in Figure 2 (b), i.e., a higher probability that unknown states will be consider as known states. This is a problem because the algorithm could decide to explore when the agent is in an unknown state, which could result in catastrophic consequences for the agent. To mitigate this problem, we reduce the width of the orange region displacing the sigmoid function in $\frac{\theta}{k}$ to the left, as shown in Equation 4. Therefore, on the other hand, the parameter k is used to displace the sigmoid function to the left, reducing in this way the probability that unknown states will be considered as known states. However, it is important to note that this probability does not completely disappear, i.e., this displacement allows also to keep the smooth transition to the right of the θ parameter. In summary, the lower the value of k is, the less aggressive the exploration of the state space will be. Therefore, lower values of k reduce the probability of damage, but may adversely affect the final performance of the algorithm. Instead, the higher the values of k are, the more similar the sigmoid function in Figure 2 (b) will be with the binary step function in Figure 2 (a). This implies a more aggressive exploration of the state space, increasing the probabilities of damages. For this reason, a correct selection of the value of the k parameter is required.

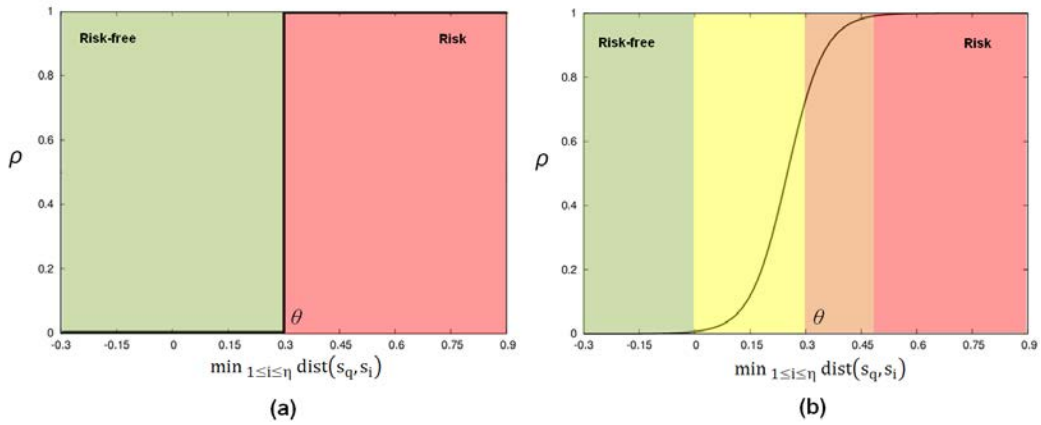

Fig. 2. (a) The discrete risk function used by the PI-SRL algorithm and (b) the continuous risk function used by the PR-SRL algorithm with $k = 6$ proposed in this paper.

It is important to note that Equation 4 does not associate risk states with catastrophic or dangerous states. Equation 4 associates risk states with unknown states, and these unknown states do not necessarily have to be catastrophic or dangerous: they are simply states in which the action to be performed is unknown. Obviously, we might reach a catastrophic state if in an unknown state a random action is performed. For this reason, the risk function in Equation 4 has a high value if s is a state far from the known space. If such is the case, it is assumed that the agent does not know what action to take in s and there should be a greater chance of asking the teacher for advice. In this way, the previously unknown state becomes a known state. In fact, the objective of the proposed exploration process is to adjust the known space and unknown space in order to explore new and improved behaviors while avoiding error states.

The risk function in Equation 4 can be seen as the probability that a state will be considered as an unknown state. Therefore, from Equation 4, the application of Probabilistic Policy Reuse to

measure the advice of a teacher in safe RL is easy, by using the risk function $\varrho^B(s)$ as a transfer function. The transfer rate depends on the safety of the learning agents: if safety is high (i.e., $\varrho^B(s)$ is low), probability to use the advice of the teacher is very low, while if safety is low (i.e., $\varrho^B(s)$ is high), such probability increases. This integration will be explained deeply later. First, Section 4 briefly summarizes the π-reuse exploration strategy.

4 THE π-REUSE EXPLORATION STRATEGY

The π-reuse strategy is an exploration strategy able to bias a new learning process with another policy [6]. Let Π_{past} be the policy to reuse and Π_{new} the new policy to be learned. The goal of π-reuse is to balance random exploration, exploitation of the past policy, Π_{past}, and exploitation of the new policy, Π_{new}, as represented in Equation 5.

$$a = \begin{cases} \Pi_{past}(s) & \text{w.p. } \psi \\ \epsilon - greedy(\Pi_{new}(s)) & \text{w.p. } (1-\psi) \end{cases} \tag{5}$$

The π-reuse strategy follows the past policy, Π_{past}, with probability ψ, and it exploits the new policy, Π_{new}, with probability of $1-\psi$. As random exploration is always required, it follows the new policy, Π_{new}, using an ϵ-greedy strategy, so random exploration is performed with probability $(1-\psi)\epsilon$. Algorithm 1 shows the procedure describing the π-reuse strategy integrated with the Q-Learning algorithm. The procedure uses as an input the past policy Π_{past}, the number of episodes P, the maximum number of steps per episode H, and the ψ parameter. An additional υ parameter is added to decay the value of ψ in each step of the learning episode. For each step h in an episode p, the algorithm selects the action a to be performed according to Equation 5 (see lines 7-8 in Algorithm 1). Then, action a is used to update $Q^{\Pi_{new}}(s,a)$ (lines 10-11). Finally, it decays the value of ψ for the next step (line 12). The procedure outputs the Q function, $Q^{\Pi_{new}}(s,a)$, and the policy, Π_{new}. The variable ψ_h keeps the value of $\upsilon^h\psi$ in each step of each episode. Therefore, in the algorithm defined in Algorithm 1, ψ and υ are parameters that regulate the transfer rate.

5 THE PR-SRL ALGORITHM

The PI-SRL algorithm uses the baseline behavior π_T in two ways resulting in two different steps. First, it uses the safe demonstrations of π_T to provide prior knowledge about the task. In this step, the algorithm builds the initial known space of the agent derived from the safe case-based policy, π_B^θ, with the purpose of mimicking π_T through π_B^θ. In the second step, PI-SRL uses π_T to support the subsequent exploration process conducted to improve the abilities of the previously-learnt π_B^θ. Instead, the PR-SRL algorithm uses the baseline behavior only to support the exploration process. Therefore, there are two main differences between the PR-SRL and the PI-SRL algorithms: (i) the PR-SRL algorithm does not perform the first step of the PI-SRL algorithm in order to properly mimic the baseline behavior and (ii) the main part of the second step (Policy Improvement) is now executed following the π-reuse exploration strategy, as defined later.

The PR-SRL algorithm does not perform the first step of the PI-SRL algorithm for three main reasons. First, one of the goals of PI-SRL is to be able to properly mimic the teacher policy by a case base. By proceeding in this way, the algorithm has proven to be safe, and the subsequent exploratory process in order to improve beyond that of the baseline behavior is optional. However, the goal of the PR-SRL is not to mimic the baseline behavior properly, but directly improve its abilities. Second, PR-SRL begins the learning process with an empty case base. Therefore, the baseline behavior π_T is invoked more frequently in early steps of the learning process than in the PI-SRL algorithm, until the case base increases the number of cases. This causes the beginning of the PR-SRL algorithm to be safer than the PI-SRL algorithm, as experimental evaluation demonstrates. And third, in the

ALGORITHM 1: π-reuse $(\Pi_{past}, P, H, \psi, v)$

Input: Π_{past}, P, H, ψ, v
Output: $Q^{\Pi_{new}}(s,a)$, Π_{new}

```
1  Initialize Q^{Π_new}(s,a) ← 0, ∀s ∈ S, a ∈ A;
2  repeat
3      Set the initial state, s, randomly;
4      h ← 1;
5      ψ_h ← ψ;
6      repeat
7          With a probability of ψ_h, a ← Π_past(s);
8          With a probability of 1 − ψ_h, a ← ε-greedy(Π_new(s)) ;
9          Receive the next state s', and reward, r_{k,h};
10         Update Q^{Π_new}(s,a), and therefore, Π_new;
11             Q^{Π_new}(s,a) ← (1 − α)Q(s,a)^{Π_new} +
                   α[r + γ max_{a'} Q^{Π_new}(s',a')]      ;
12         ψ_{h+1} ← v^h ψ;
13         s ← s';
14         h ← h + 1;
15     until h < H;
16     p ← p + 1;
17 until p < P;
```

PR-SRL algorithm, all the cases added to the case base are derived from the exploratory process supported by the baseline behavior in order to improve its abilities. This could avoid introducing a bias in the learning process caused by trying to imitate firstly the baseline behavior.

Algorithm 2 shows Safe π-reuse, a version of the π-reuse algorithm to incorporate the teacher advice in the exploration process. The main modifications are:

- the past policy Π_{past} is replaced by the baseline behavior π_T
- the new policy to be learned Π_{new} is replaced by the case base policy π_B^θ
- the parameter ψ is replaced by $\varrho^B(s)$
- no ϵ-greedy strategy is used, because actions are continuous. The random Gaussian noise, as defined in Section 2.4 is used instead to generate exploratory actions

The algorithm builds a case for each step of an episode. For each new state s_h, the closest case $< s, a, V(s) > \in B$ is computed using the Euclidean distance metric defined in Equation 1 (see line 3 in Algorithm 2). At this point, the π-reuse strategy is followed, using the $\varrho^B(s)$ function as a transfer probability (ψ parameter in Equation 5): with a probability of $\varrho^B(s)$ the policy of the baseline behavior π_T is followed, while with a probability of $1 - \varrho^B(s)$, the action suggested by current case base policy is executed. Therefore, in areas far from the known states, the probability to use the teacher advice is very high, while this advice is rarely used in known areas.

If the algorithm follows the baseline behavior the action a_h performed is suggested by the baseline behavior π_T which defines safe behavior, and a new case $< s_h, a_h, 0 >$ is built (line 5). In this case, the state s_h is considered an unknown state. If the algorithm exploits the new policy the action a_h performed is computed using Equation 2 (line 6). In this case, the new case $< s, a_h, V(s) >$ is built replacing the action a corresponding to the closest case in $< s, a, V(s) > \in B$, with the new action a_h resulting from the application of random Gaussian noise to a in the Equation 2. Thus,

ALGORITHM 2: Safe π-reuse $(\pi_T, H, B, \sigma, \theta, k)$

Input: π_T, H, B, σ, θ, k
Output: $listCasesEpisode$, $totalRwEpisode$

1 Initialize $listCasesEpisode \leftarrow \emptyset$, $totalRwEpisode \leftarrow 0$, $h \leftarrow 1$, initial state, s_h
2 **repeat**
3 Compute the case $< s, a, V(s) > \in B$ closest to the current state s_h;
4 $\varrho^B(s_h) \leftarrow 1 - \frac{1}{1+e^{\frac{k}{\theta}((\min_{1\leq j\leq \eta} d(s_h, s_j) - \frac{\theta}{k}) - \theta)}}$;
5 With a probability of $\varrho^B(s_h)$: $a_h \leftarrow \pi_T(s_h)$, $c^{new} \leftarrow (s_h, a_h, 0)$;
6 With a probability of $1 - \varrho^B(s_h)$: $a_h \leftarrow rnd\_gauss(\pi_B(s_h), \sigma)$, $c^{new} \leftarrow (s, a_h, V(s))$;
7 Execute a_h and receive the next state s'_h, and reward, r_{s_h, a_h};
8 $totalRwEpisode \leftarrow totalRwEpisode + r(s_h, a_h)$;
9 $listCasesEpisode \leftarrow listCasesEpisode \cup c^{new}$;
10 $s_h \leftarrow s'_h$;
11 $h \leftarrow h + 1$;
12 **until** $h < H$;

the algorithm only produces smooth changes in the cases of B where $a_h \sim a$. In this case, the state s_h is considered a known state. Finally, the reward obtained in the episode is accumulated, where $r(s_h, a_h)$ is the immediate reward obtained when action a_h is performed in state s_h (line 8) and the new case is added to the list of cases (line 9).

The PR-SRL algorithm is depicted in Algorithm 3. The algorithm is composed of three steps performed in each episode.

- **(a) Case Generation**. The algorithm uses the π-reuse exploration strategy in Algorithm 2 to build the cases in the episode (see line 3 in Algorithm 3).

- **(b) Computing the state-value function for the unknown states**. In this step, the state-value function of the states considered to be unknown in the π-reuse exploration is computed. In the previous step (line 3), the state-value function for these states is set to 0. The algorithm proceeds in a manner similar to the first-visit MC algorithm [28]. In this case, the return for each considered unknown state s_i is computed, but not averaged since only one episode is considered (lines 6-7). The return for each s_i is computed, taking into account the first visit of the state s_i in the episode (each occurrence of a state in an episode is called a visit to s_i), although the state s_i could appear multiple times in the rest of the episode. Anyway, it can be considered as an initialization procedure, since the return for each s_i will be adjusted on subsequent visits to these states as described in step (c).

- **(c) Updating the cases in *B* using experience gathered**. Updates in B are made with the cases gathered from episodes with a cumulative reward similar to that of the best episode found to that point using the threshold Θ (line 10). Thus, the updates in B will be made with the cases gathered from episodes with a similar quality to the best episode found so far. In this step, two types of updates appear, namely, replacements and additions of new cases. Again, the algorithm iterates for each case $c_i = (s_i, a_i, V(s_i)) \in listCasesEpisode$ (line 12). If s_i was considered a known state during the π-reuse exploration (line 13), we compute the case $< s_i, a, V(s_i) > \in B$ corresponding to the state s_i (line 14). One should note that the case $c_i = (s_i, a_i, V(s_i)) \in listCasesEpisode$ was built in line 3 within the Algorithm 2, replacing the action a corresponding to the case $< s_i, a, V(s_i) > \in B$ with the new action a_i and resulting from the application of random Gaussian noise to the action a by the Equation 2. Then, the temporal distance (TD) error δ is computed (line 15). If $\delta > 0$,

ALGORITHM 3: PR-SRL($\eta, \pi_T, \Theta, \sigma, \theta, k, P, H$)

Input: η,π_T,Θ,σ, θ, P, H
Output: B

1 Initialize $maxTotalRwEpisode \leftarrow 0$, $B \leftarrow \emptyset$;
2 **repeat**
  /* **(a) Case generation** */
3 $listCasesEpisode, totalRwEpisode \leftarrow \pi$-reuse($\pi_T$,$H$,$B$,$\sigma$,$\theta$,$k$);
  /* **(b) Computing the state-value function for the unknown states** */
4 **foreach** $c_i \in listCasesEpisode$ **do**
5 **if** s_i *was considered an* unknown state **then**
6 $return(s_i) \leftarrow \sum_{j=n}^{k} \gamma^{j-n} r(s_j, a_j)$;
7 $V(s_i) \leftarrow return(s_i)$;
8 **end**
9 **end**
  /* **(c) Updating the cases in B using the experience gathered** */
10 **if** $totalRwEpisode > (maxTotalRwEpisode - \Theta)$ **then**
11 $maxTotalRwEpisode \leftarrow \boldsymbol{max}(maxTotalRwEpisode, totalRwEpisode)$;
12 **foreach** $c_i =< s_i, a_i, V(s_i) > \in listCasesEpisode$ **do**
13 **if** s_i *was considered a* known state **then**
14 Compute the case $< s_i, a, V(s_i) > \in B$ corresponding to the state s_i;
15 Compute $\delta \leftarrow r(s_i, a_i) + \gamma V(s_{i+1}) - V(s_i)$;
16 **if** $\delta > 0$ **then**
17 Replace $< s_i, a, V(s_i) > \in B$ with $< s_i, a_i, V(s_i) > \in listCasesEpisode$;
18 $V(s_i) \leftarrow V(s_i) + \alpha\delta$;
19 **end**
20 **end**
21 **else**
22 $B \leftarrow B \cup c_i$;
23 **end**
24 **end**
25 **end**
26 **if** $\|B\| > \eta$ **then**
27 Remove the $\eta - \|B\|$ least-frequently-used cases in B;
28 **end**
29 $p \leftarrow p + 1$
30 **until** $p < P$;

performing the action a_i results in a positive change for the value of a state. The action, in turn, could potentially lead to a higher return and, thus, to a better policy. Van Hasselt and Wiering [33] also update the value function using only the actions that potentially lead to a higher return. If the TD error δ is positive, a_i is considered to be a good selection and is reinforced. In the algorithm, this reinforcement is carried out by updating the output of the case $< s_i, a, V(s_i) > \in B$ at a_i (line 17). Therefore, an update to the case-base only occurs when the TD error is positive. Additionally, it prevents the degradation of B, ensuring that replacements are made only when an action can

potentially lead to a higher $V(s_i)$. If, instead, s_i was considered an unknown state, the case c_i is added to B (line 22). In this way, the unknown state s_i becomes a known state. Finally, the algorithm removes cases from B if necessary (line 25). Therefore, the algorithm can also "forget" ineffectual known states.

6 EXPERIMENTAL RESULTS

This section reports the experimental results of using the PR-SRL algorithm to learn policies in two different domains: the generalized helicopter hovering domain [23], and the business simulator SIMBA [3]. We propose to learn a near-optimal policy avoiding the failures, i.e., the helicopter crashes, and company bankruptcies during the learning phase. The results of PR-SRL in these domains are compared with the results of PI-SRL, which has proven to have less failures (i.e., collisions, bankruptcies) than other Risk-sensitive approaches [9], and the evolutionary RL approach, selected winner of the helicopter domain in the 2009 RL Competition [20]. In the evolutionary RL approach, the weights of neural networks are evolved by inserting several baseline behaviors into the initial population. One of these baseline behaviors is exactly the same as that used in PR-SRL. This makes the comparison of performances as fair as possible, but taking into account that each of the different techniques makes its own use of the baseline behaviors. Firstly, Section 6.1 proposes a guideline to set the parameters of PR-SRL.

6.1 Parameter Setting Design

One of the main difficulties of applying the PR-SRL algorithm to a given problem is deciding on an appropriate set of parameter values. It is important to be aware of the fact that doing any sort of parameter tuning by running experiments has to be avoided or, at least, such experiments have to be conducted as safely as possible, since it would cause failures while looking for parameters that avoid failures. For this reason, in this section a solid perspective is given on the automatic definition of these parameters. The guidelines to set the parameter value for threshold θ, the update threshold Θ, the maximum number of cases η, and the risk parameter σ have been previously proposed by García and Fernández for the PI-SRL algorithm [9]. We are using such guidelines for PR-SRL as well. Additionally, in this section we also propose a guideline to set the new parameter k.

In respect to the parameters θ and η, we run the baseline behavior π_T to compute its values. The value of the parameter θ is established by computing the mean distance between states during an execution of π_T. Instead, η is set by computing the distances between the states in different runs of π_T [9]. Therefore, the values for θ and η are computed by running experiments, but such experiments are conducted as safely as possible using the baseline behavior π_T. In respect to the update threshold Θ, it is used to determine how good an episode must be with respect to the best episode obtained, since only the best episodes are used to update the case-base B. Therefore, it is not related to risk: its value should affect the final performance of the algorithm but does not compromise the safety of the agent. In any case, we propose to use a fixed value for it: $\Theta = 5\%$ of the cumulative reward of the best episode obtained. Such a value has been proven successfully in several domains [9].

Once the parameters θ, η and Θ are set, we focus on the parameters σ and k. Both parameters are used to determine the level of risk assumed during the exploratory process. Small values of these parameters make the exploration process too careful, while high values make the exploration more aggressive, increasing the probability of failure. Therefore, intuitively, it is advisable beginning with a small value of σ (e.g., $\sigma = 9 \times 10^{-7}$). Then, for this value, it is advisable to first test a small k value (e.g., $k = 1$) and then increase it iteratively up to a maximum (e.g., $k = 12$) or until an accurate policy is obtained. If the maximum is reached and such policy is not obtained, σ is increased (e.g., $\sigma = 9 \times 10^{-6}$), k is reset to its initial small value, and the process is repeated. Proceeding in this

way, the level of risk assumed increases from run to run even more smoothly than just increasing the value of the parameter as in previous works [9]. Thus, it is possible to adjust the value of σ and k in the safest possible way.

Although σ should be increased iteratively, let us in this paper focus on a single value of σ both in the helicopter domain ($\sigma = 9 \times 10^{-3}$), and in the business simulator ($\sigma = 9 \times 10^{-1}$). Such values allow PI-SRL to obtain the highest cumulative reward according to our previous work [9]. The purpose of fixing these values is to demonstrate that our new PR-SRL algorithm is able to achieve a similar cumulative reward than the best obtained by PI-SRL, while reducing its number of failures or even removing them completely. Additionally, focusing on a single σ value for each domain facilitates the analysis of the new parameter k. In any case, in experiments of Sections 6.2 and 6.3, the parameter k is gradually incremented from $k = 1$ to $k = 12$ as proposed.

Finally, it is important to note that these guidelines allow the value of the parameters to be set safely and, in the absence of knowledge of the domain, they have proven to be a suitable set of heuristics tested successfully in a wide variety of domains. Obviously, it cannot be guaranteed that in all cases these guidelines lead to an exploration completely free of failures, but they empirically produce a drastic reduction in the number of them as will be demonstrated in Sections 6.2 and 6.3.

6.2 Helicopter Domain

First, we consider the helicopter domain from the RL competition [23]. In this domain, the goal is to make the helicopter hover as closely as possible to a defined position for the established duration of an episode. This domain represents a challenging task for two main reasons. First, both the state and action spaces are high-dimensional and continuous (the state space is 12-dimensional and the action space is 4-dimensional). Second, it is a generalized domain where the wind factor modifies the domain behavior. A helicopter episode is composed of 6000 steps, but an episode could end prematurely if the helicopter crashes. At each step, the agent receives a reward that is the squared difference between the current state and the position in which the helicopter wishes to hover, while crashing results in a large negative reward [16]. In this case, we use the sub-optimal behavior provided by the software of the competition as the baseline behavior, π_T, with a mean cumulative reward per episode of -78035.93. Both in PR-SRL and PI-SRL, θ, η, σ and Θ were computed following the procedure described in Section 6.1 with values of 0.3, 49735, 9×10^{-3}, and 5% respectively.

6.2.1 Eliminating the First Step of the PI-SRL algorithm. One of the contributions of PR-SRL over PI-SRL is that the first learning step, *Modeling the Baseline Behavior* is eliminated, so the initial case base used is empty (i.e., at the beginning of the exploration process all the states are unknown, $\Upsilon = S$). To evaluate such effect from the rest of the contributions presented in this work, we have executed the PI-SRL algorithm both with and without that first step. Figure 3 shows the performance of the second step of the PI-SRL algorithm starting with an empty case base (i.e., without the previous execution of the first step), and the performance of the second step starting with a case base that mimics the base line behavior (i.e., with the previous execution of the first step). The latter is the normal operation of the PI-SRL algorithm.

In the early steps of the learning process, PI-SRL with an empty case base starts below PI-SRL and progressively, as the learning process proceeds, it reaches a similar performance. At the beginning of the learning process, the PI-SRL algorithm with an empty case base only executes baseline behavior advices, maintaining the performance below the normal PI-SRL. But progressively, while the case base grows, it decreases the baseline behavior advices, and increases the exploratory steps, while increasing the performance of the algorithm. Additionally, both versions of the algorithm obtain a similar number of failures. Therefore, the main difference between the two versions of the

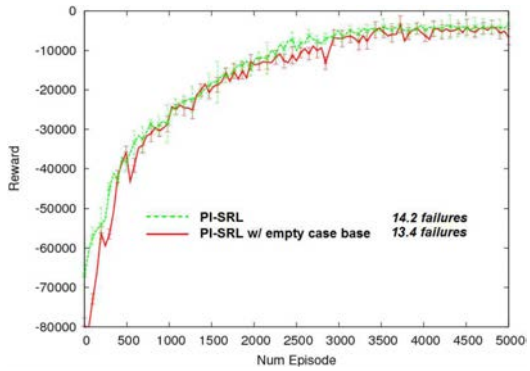

Fig. 3. Mean cumulative reward per episode obtained by the PI-SRL with and without the execution of the first step *Modeling the Baseline Behavior*. The means have been computed from 10 different executions.

algorithm is that PI-SRL with an empty case base uses the early steps of the learning process as the first step of the regular PI-SRL algorithm, and if the goal is not to obtain a case base that mimics the baseline behavior, the first step can be skipped. Next, the effect of using Policy Reuse and the continuous risk function is analyzed.

6.2.2 Using Policy Reuse with a Continuous Risk Function. In this section we evaluate the effect of using a continuous risk function. We have applied three different risk functions as defined in Figure 2.

- The discrete risk function in Equation 3, as it was defined in previous works [9].
- The continuous risk function, as defined in Equation 4. We call this function the displaced risk function
- The same as before, but without the displacement over the θ parameter by θ/k , i.e:

$$\varrho^B(s) = 1 - \frac{1}{1 + e^{\frac{k}{\theta}((\min_{1 \leq j \leq \eta} d(s, s_j)) - \theta)}} \tag{6}$$

As will be shown later, the second one is the one that obtains better results, but we compare against the other because such comparison allows us a better understanding of why the safe behaviors are obtained. Figure 4 (a) and (b) plot, respectively, the non-displaced and the displaced risk functions with respect to the discrete one. In each plot, we show the risk function for different values of k. We can see that, depending on such value, the risk function becomes closer to the step function (higher values), or more flattened (lower values). Let's think about the risk function in terms of a probability distribution of the unknown states. The binary step function perfectly separates the region of the known states from the region of the unknown states: if $\min_{1 \leq i \leq \eta} d(s_q, s_i) \leq \theta$, the probability to be an unknown states is 0; for values higher than θ, such probability is 1 (for definition of known and unknown states see Definition 2.3. Therefore, such function can be considered a perfect classifier. Let's focus now on the case of the continuous risk function with $k = 6$. The yellow area represents the states, s_q, that are located in the region of the known states but that are "classified" as unknown. They are the false positives. The blue area represents the false negatives, i.e. the states that are located in the area of the unknown states but are classified as positive. For the risk functions with $k = 3$, the same regions appear, although the areas of false positives and false negatives are bigger, which will have consequences to be shown later.

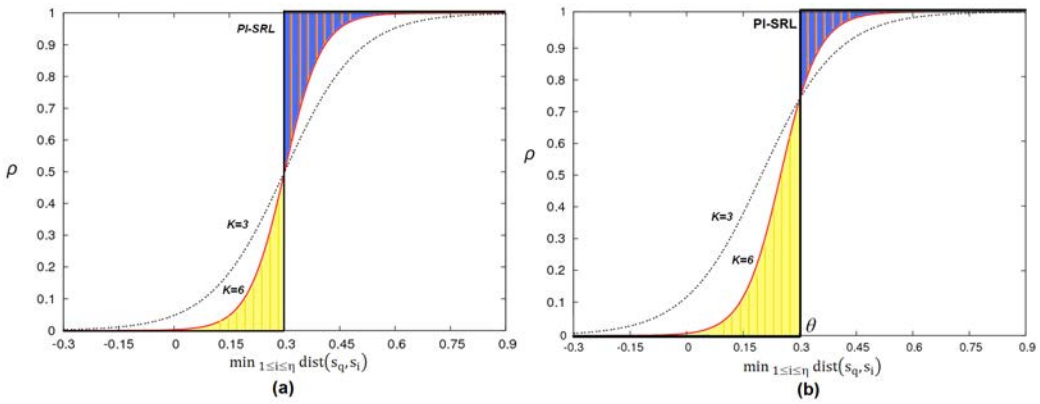

Fig. 4. (a) Nondisplaced sigmoid functions for different values of k, and the discrete risk function of PI-SRL. (b) Displaced sigmoid functions for different values of k, and the discrete risk function of PI-SRL.

From the exploration point of view, the yellow area is an area where the agent acts more carefully than needed: the agent is in a known state, and follows the baseline behavior advice. The blue area is the really risky region, because the agent is in an unknown state but it explores randomly.

Figure 5 (a) shows the states visited by the learning agent when using the different risk functions. Green bars represent the number of times that the agent explored known states, while red bars represents the number of times that the agent followed the teacher advice in unknown states. Both situations are assumed to be correct. As in previous figure, yellow and blue colors represent the error areas. PI-SRL visited 5333.31 known states and 666.68 unknown states. Using the continuous risk function, PR-SRL $k = 12$ without the displacement visited 5282.7 known states and 717.3 unknown states. From the 5282.7 known states, 482.91 states were considered unknown states by the continuous risk function (false positives), so the teacher advice was followed. This results in a less aggressive exploration of the state space. Instead, from the 717.3 unknown states, 122.31 states are considered known states by the continuous risk function (false negatives). This is a problem since the algorithm decides to explore when the agent is in an unknown state, which could result in catastrophic consequences. For this reason, the sigmoid function is displaced to the left in $\frac{\theta}{k}$. Proceeding in this way, Figure 5 (a) shows that PR-SRL $k = 12$ increases the number of known states considered unknown (739.36), but reduces the number of unknown states considered known states (46.55). The same interpretation can be applied to PR-SRL $k = 6$ with and without displacement of $\frac{\theta}{k}$. In summary, using Equation 4, lower values of k imply a less aggressive exploration of the state space, and a lower probability of false negatives.

Figure 5 (b) presents the number of steps per episode executed by the baseline behavior π_T during the execution of the second step of the PI-SRL algorithm, and during the execution of the PR-SRL algorithm (Algorithm 3) for different values of k. It shows the first 50 episodes of the learning processes. In the case of the PR-SRL algorithm, at the beginning of the learning process with an empty case-base B, all steps are performed using the baseline behavior π_T. As the learning process proceeds, new cases are added to B and the number of required executions of the baseline behavior is reduced. Furthermore, the lower the value of the k parameter is (i.e., the wider the sigmoid function), the greater the number of baseline behavior advices during the learning process (i.e., the exploration process is less aggressive). Instead, the PI-SRL algorithm begins the learning process with a learned case base trying to mimic the baseline behavior. For this reason, the number

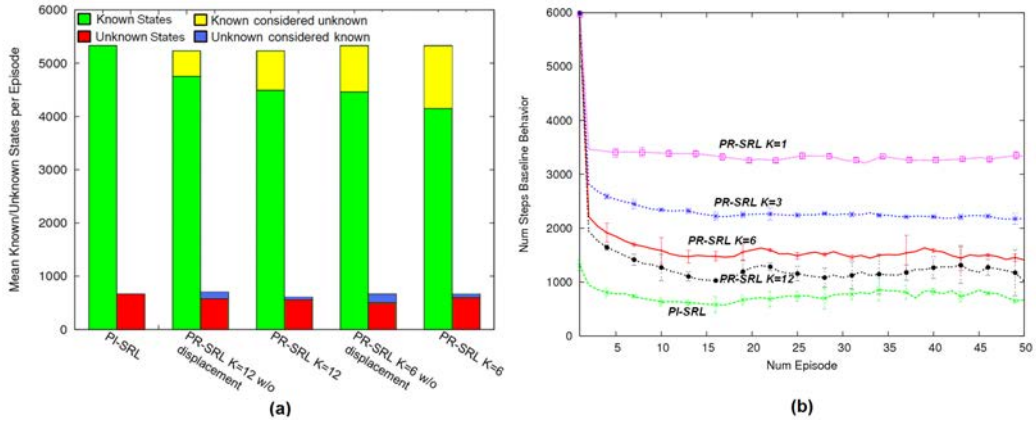

Fig. 5. (a) Mean number of known/unknown states per episode of the PI-SRL and PR-SRL for different values of k for sigmoid functions with and without displacement. (b) Evolution of the mean number of steps executed by the Baseline Behavior in the second step of the PI-SRL algorithm, and in the PR-SRL algorithm for different values of the k parameter. The means have been computed from 10 different executions.

of required executions of the baseline behavior is almost constant throughout the whole learning process.

Figure 6 (a) shows the mean number of failures and cumulative reward over 5000 episodes for different approaches. The data has been computed from 10 independent executions of each approach. In particular, Figure 6 (a) shows the performance of an Evolutionary RL algorithm (pink diamond), the performance of the PI-SRL algorithm (red triangle), the performance for PR-SRL with a non-displaced risk function (green squares), and, finally, PR-SRL with a displaced risk function (blue circles). In respect to PR-SRL, Figure 6 (a) shows the performance for different k values incrementally tested: $k = 1$, $k = 3$, $k = 6$ and $k = 12$. In PR-SRL w/o displacement (green squares), there is a probability that known states will be considered as unknown states (false positives), which implies the agent acts more carefully than needed. In fact, the lower the value of the parameter k is, the more carefully the exploration will be performed, and the lower the cumulative reward obtained will be. Additionally, there is a high probability that unknown states will be considered as known states (false negatives), which increases the probability of failures. For this reason, PR-SRL without displacement generates failures. A higher value of k, produces a more aggressive exploration, since the continuous risk function looks like the discrete function used by PI-SRL. However, PR-SRL with $k = 6$ obtains a lower number of failures and a similar performance to PI-SRL (red triangle), proving that using the continuous risk function is better than using the binary step function.

In cases of PR-SRL using the displaced risk function (blue circles), the probability that known states will be considered as unknown states (false positives) is increased, and the probability that unknown states will be considered as known states (false negatives) is decreased w.r.t. PR-SRL without displacement (green squares). For this reason, for each value of k, PR-SRL slightly reduces the cumulative reward obtained in comparison to PR-SRL without displacement, but it also reduces the number of failures obtained. In fact, the decrement in false negatives makes the PR-SRL algorithm with values $k = 1$, $k = 3$ and $k = 6$ not to generate failures. In PR-SRL $k = 12$ the sigmoid function begins to look like the binary step function used by the PI-SRL algorithm, and the failures appear. Therefore, using PR-SRL $k = 6$, we are able to completely avoid the visit to undesirable situations, while maintaining the final performance achieved by PI-SRL and the

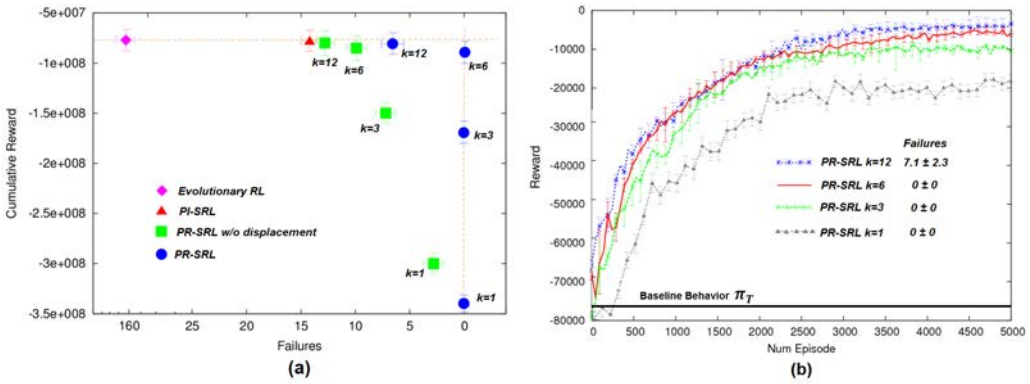

Fig. 6. (a) Mean number of failures and cumulative reward obtained by PI-SRL and by PR-SRL w/ and w/o displacement using different values of k. (b) Mean cumulative reward per episode obtained by PR-SRL w/ displacement using different values of k. The means have been computed from 10 different executions.

Evolutionary RL algorithm. It is important to note that given the good performance of the algorithm with $k = 6$, it could be decided not to execute it with $k = 12$, preventing possible failures of a more aggressive exploration process. Finally, Figure 6 (b) shows the evolution of the learning processes corresponding to PR-SRL with different values (blue circles) in Figure 6 (a). It also shows the performance of the baseline behavior (horizontal black line). Figure 6 (b) shows that from the beginning of the learning processes, the use of low k values and a less aggressive exploration affect the final performance of the algorithm.

Finally, we have performed additional experiments in the helicopter control task using a poor and a very poor policy as baseline behaviors. Figure 7 (a) shows the performance of these policies used as baseline behaviors. The continuous black line indicates the performance of the baseline behavior used in previous experiments. As described previously, this baseline behavior is a safe behavior free of failures. The continuous red line indicates the performance of the poor policy, and the dashed green line shows the performance of the very poor policy. Both the poor and the very poor policy are stochastic policies which perform under the safe baseline behavior. Additionally, the poor policy causes helicopter crashes occasionally, but the very poor policy causes the helicopter to crash constantly. Finally, Figure 7 (a) shows a horizontal blue line corresponding to the performance of the best policy obtained by the PI-SRL algorithm in this domain.

On the other hand, Figure 7 (b) shows the performance of PI-SRL and PR-SRL with $k = 6$ using the poor and the very poor policies as baseline behaviors. Figure 7 (b) indicates that with the use of the poor policy as baseline behavior, both PI-SRL and PR-SRL are able to learn a near-optimal policy as well as when a policy free of failures is used. However, Figure 7 (b) also shows that with the use of the very poor policy the performance of both algorithms is greatly decreased. Finally, PR-SRL is not able to completely avoid the failures due to the use of non-safe baseline behaviors, but it does reduce drastically the number of them in comparison with PI-SRL. Additionally, PR-SRL obtains fewer failures than the poor and very poor policy. In fact, the number of failures is higher at the beginning of the learning processes. As the learning process proceeds, the different policies move away from the error space, and the number of failures decreases.

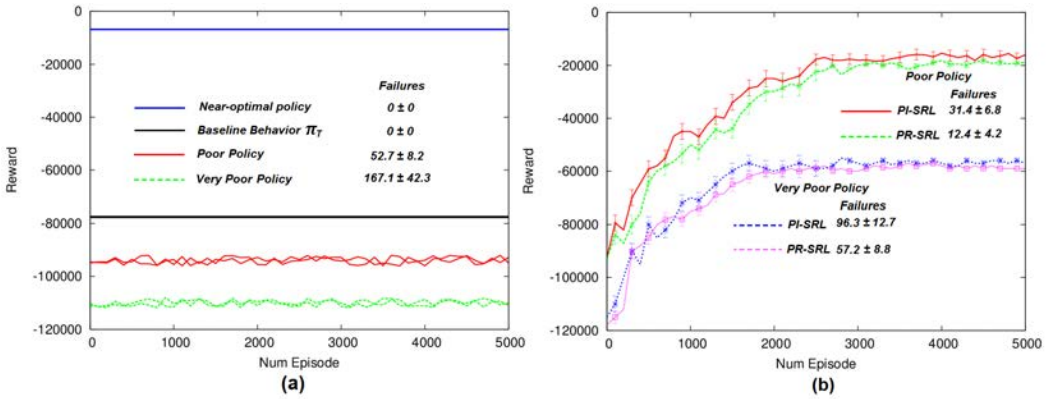

Fig. 7. (a) The performance of different baseline behaviors. (b) The performance of PR-SRL and PI-SRL using different baseline behaviors.

6.3 SIMBA

Business simulators are a powerful tool to improve management decision-making processes. An example is SIMBA (*SIMulator for Business Administration*) [3]. SIMBA is a competition simulator, since agents can compete against other agents each one managing a different virtual company. In the experiments performed here, the learning agent competes against five hand-coded agents [3]. Decision making in SIMBA is an episodic task where decisions are sequentially taken. To make a business decision requires studying the state and setting 10 continuous decision variables (selling price, advertising expenses, etc.), and the decisions are taken after the study of a state composed of 12 continuous variables (material costs, financial expenses, economic productivity,. . .) [3]. Therefore, the search space in SIMBA is also continuous and Euclidean. Each episode is composed of 52 steps, and it can end prematurely if the company is bankrupted (company losses are higher than 10% of net assets). SIMBA constitutes another domain of application for our safe exploration method. Therefore, our proposal is not only applicable to robotic domains, but to all kinds of tasks where a safe exploration of the state and action space is required. In these experiments, both in PR-SRL and PI-SRL, θ, η, σ and Θ were computed following the procedure described in Section 6.1 with values of 100, 503, 9×10^1, and 5% respectively.

Figure 8 (a) shows the mean number of failures and cumulative reward over 100 episodes for an Evolutionary RL algorithm (pink diamond), the PI-SRL algorithm (red triangle), PR-SRL with a non-displaced risk function (green squares) and, finally, PR-SRL with a displaced risk function (blue circles). In respect to PR-SRL, Figure 8 (a) shows the performance for different incremental values of k, in particular, $k = 1$, $k = 3$, $k = 6$ and $k = 12$.

Figure 8 (a) allows the same conclusions that are in Figure 6 (a). On one hand, small k values affect the final performance of the algorithm, although the obtained behavior is safer. On the other hand, the use of a displaced risk function allows a reduction in the number of failures. Thus, for example, if we focus on the blue circles, PR-SRL with $k = 1$, $k = 3$ and $k = 6$ does not produce failures, while a value of $k = 12$ begins to produce them. Additionally, it is interesting to note that once again PR-SRL with $k = 6$ performs similarly to PI-SRL, but, in contrast, it does not produce failures. It seems that values above $k = 6$ performs similarly to PI-SRL, while values bellow $k = 6$ performs an exploration too conservative. Therefore, the experiments demonstrate that $k = 6$ is a good intermediate value that allows one to obtain a similar performance to PI-SRL, while reducing the number of failures or even removing them completely. Once again, given the good performance

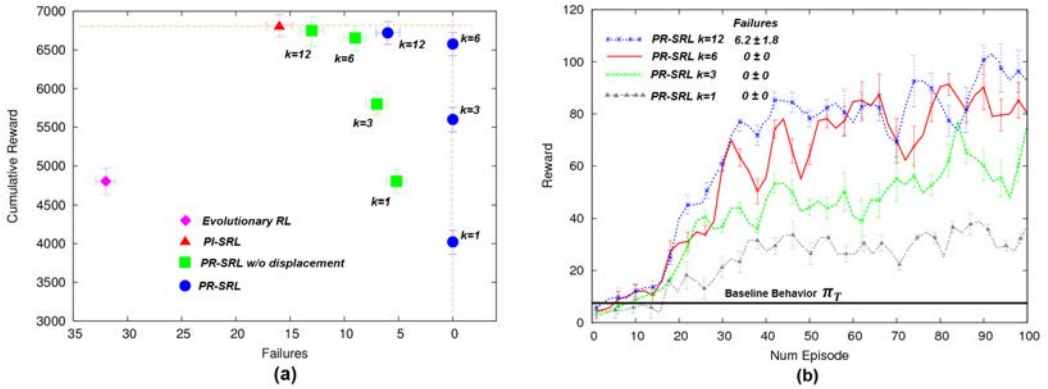

Fig. 8. (a) Mean number of failures and cumulative reward obtained by PI-SRL and by PR-SRL w/ and w/o displacement using different values of k. (b) Mean cumulative reward per episode obtained by PR-SRL w/ displacement using different values of k. The means have been computed from 10 different executions.

of the algorithm with $k = 6$ it could be decided not to execute it with $k = 12$, preventing a more aggressive exploration and possible failures. Finally, Figure 8 (b) shows the evolution of learning processes corresponding to the blue points in Figure 8 (a). Figure 8 (b) shows how high values perform a more aggressive exploration from the beginning of the learning process, which also implies a higher probability of failure.

7 CONCLUSIONS

In this work, PR-SRL, an algorithm for safe reinforcement learning in high-risk tasks, is described. The helicopter hovering task and the business simulator SIMBA have been used to illustrate the performance of the algorithm. Next we summarize the main conclusions found in this paper:

(i) The continuous risk function allows a smooth transition between the known and unknown states. In contrast to the discrete risk function used in PI-SRL [9], the continuous risk function allows one to model the risk concept more naturally. This function progressively increases the probability that a state will be considered as unknown as we approach the frontier marked by the parameter value θ. Therefore, the value of our risk function is not only 0 or 1, i.e., it is not just a binary decision on whether or not to provide an advice as the discrete functions do. It has infinite intermediate values in the range $[0, 1]$. In this way, it represents more naturally the way in which humans perceive the risk. Humans are able to distinguish different levels of risk and they act in one way or another according to this perceived level. In our case, the higher the value of the risk function is, the higher the probability of using the advice of the teacher will be.

(ii) The continuous risk function increases the false negatives. Section 6.2.2 demonstrates that the use of the continuous risk function increases the false negatives, i.e., the states that are located in the area of the unknown states but are classified as known states. These false negatives could lead the agent to visit undesirable situations because the agent is in an unknown state but it explores randomly. Mitigating these difficulties, a displacement of the sigmoid function is introduced. In this way, we reduce the region of the state space in which the false negatives are produced. The experiments in Section 6.2.2 demonstrate that it reduces drastically the number of false negatives, and, hence, the probability of visiting error states. This displacement also increases the probability that known states will be considered as unknown states, i.e., the agents act more carefully than needed. However, proceeding in this way, the number of failures is reduced when compared with

non-displaced continuous and discrete risk functions, while maintaining a similar performance (Section 6.2.2 and 6.3).

(iii) Integration of teacher advice by π-reuse. The previous risk function allows the integration of teacher advice in the π-reuse algorithm. The parameter ϕ in Algorithm 1 is replaced by the risk function $\varrho^B(s)$ in Algorithm 2. In this way, ϱ^B is used as a transfer probability: with a probability of $\varrho^B(s)$ the policy of the baseline behavior π_T is followed, while with a probability of $1 - \varrho^B(s)$, the action suggested by the current case base policy is executed. The experiments in Section 6 demonstrate that such a way of incorporating teacher advice in Algorithm 3 produces an efficient and safe exploration of the state and action spaces.

(iv) Elimination of the first step, Modeling the Baseline Behavior step, of the PI-SRL algorithm. Section 6.2.1 demonstrates that it is not necessary to start the exploration of the space with a case base which mimics the behavior of the teacher π_T in order to improve this behavior. Instead, PR-SRL begins the learning process with an empty case base. This means that the baseline behavior π_T is invoked more frequently at the beginning of the learning process than in the PI-SRL algorithm, until the case base increases the number of cases. This causes the beginning of the PR-SRL algorithm be safer than the PI-SRL algorithm, as experimental evaluation demonstrates.

(v) Completely avoiding the visit to undesirable situations during learning. The main contribution of this paper is an algorithm able to completely avoid failures during the exploration of the space. In our case, such exploration is supported by the safe baseline behavior described in Section 2.3. An efficient and safe exploration process is required in tasks such as robotic tasks in which the visit to undesirable situations has catastrophic consequences. In these tasks, we cannot spend time randomly exploring irrelevant regions of the space. Additionally, we want to preserve the integrity of the robot during the learning process. The experiments in Section 6 demonstrate that the proposed algorithm does not produce failures and it is able to achieve near-optimal policies. Additionally, the proposed risk metric is easily generalizable to any risky domain.

(vi) Limitations and domain applicability restrictions of the proposed algorithm. In what follows, we discuss the applicability of the method, allowing the reader to more clearly understand the scenarios in which the proposed PR-SRL approach may be applicable. First, the applicability of the proposal is only advisable in risky domains where it is crucial to protect the safety of the agent, the learning system, or other entities throughout the training process. In domains where safety is not a requirement, other forms of exploration are also a good option [11, 27]. Second, the proposed method requires the presence of a baseline behavior (a teacher) that safely demonstrates the task to be learned. However, such presence is not guaranteed in all domains. Third, we assume some degree of smoothness in the dynamic behavior of the system. In particular, such smoothness is based on two common assumptions in RL derived from the following generalization principles [14]: (i) we assume that similar actions in similar states should produce similar effects and, (ii) we assume that nearby states have similar optimal actions. Finally, as explained earlier, the present study uses Euclidean distance as a similarity metric, as it has been proven successful in the proposed domains with Euclidean space. In fact, the applicability of the proposal is restricted to domains in which a distance metric can be defined. However, in certain environments, these assumptions may not hold.

In summary, PR-SRL algorithm represents an interesting, novel, and promising exploration process which is able to completely avoid the visits to undesirable situations while obtaining near-optimal policies. One future work would be the deployment of the algorithm in real environments, where the performance of discrete and continuous risks functions will be compared. Finally, another interesting future work would be to adapt the level of risk during learning, e.g., σ may be higher when you are far away from unknown states than when you are close to them.

ACKNOWLEDGMENTS

This paper has been partially supported by the Spanish Ministerio de Economía y Competitividad TIN2015-65686-C5-1-R and the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement No. 730086 (ERGO). Javier García is partially supported by the Comunidad de Madrid (Spain) funds under the project 2016-T2/TIC-1712.

REFERENCES

[1] Agnar Aamodt and Enric Plaza. 1994. Case-based reasoning; Foundational issues, methodological variations, and system approaches. *AI COMMUNICATIONS* 7, 1 (1994), 39–59.

[2] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. 2009. A Survey of Robot Learning from Demonstration. *Robot. Auton. Syst.* 57, 5 (May 2009), 469–483.

[3] Fernando Borrajo, Yolanda Bueno, Isidro de Pablo, Begoña Santos, Fernando Fernández, Javier García, and Ismael Sagredo. 2010. SIMBA: A Simulator for Business Education and Research. *Decision Support Systems* 48, 3 (2010), 498–509.

[4] Yin Cheng and Weidong Zhang. 2018. Concise deep reinforcement learning obstacle avoidance for underactuated unmanned marine vessels. *Neurocomputing* 272 (2018), 63–73.

[5] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. 2013. A Survey on Policy Search for Robotics. *Foundations and Trends in Robotics* 2, 1-2 (2013), 1–142.

[6] Fernando Fernández and Manuela Veloso. 2006. Probabilistic Policy Reuse in a Reinforcement Learning Agent. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '06)*. ACM, New York, NY, USA, 720–727.

[7] Fernando Fernández and Manuela M. Veloso. 2013. Learning domain structure through probabilistic policy reuse in reinforcement learning. *Progress in Artificial Intelligence* 2, 1 (2013), 13–27.

[8] Javier García, Fernando Borrajo, and Fernando Fernández. 2012. Reinforcement Learning for Decision-Making in a Business Simulator. *International Journal of Information Technology & Decision Making (IJITDM)* 11, 05 (2012), 935–960.

[9] Javier García and Fernando Fernández. 2012. Safe Exploration of the State and Action Spaces In Reinforcement Learning. *Journal of Artificial Intelligence Research (JAIR)* 45 (2012), 515–564.

[10] Peter Geibel and Fritz Wysotzki. 2005. Risk-Sensitive Reinforcement Learning Applied to Control under Constraints. *Journal of Artificial Intelligence* 24 (2005), 81–108.

[11] Todd Hester and Peter Stone. 2013. TEXPLORE: real-time sample-efficient reinforcement learning for robots. *Machine Learning* 90, 3 (01 Mar 2013), 385–429.

[12] Jonathan Ho and Stefano Ermon. 2016. Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Curran Associates Inc., USA, 4565–4573.

[13] Bing-Qiang Huang, Guang-Yi Cao, and Min Guo. 2005. Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, Vol. 1. IEEE, 85–89.

[14] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. 1996. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research* 4 (1996), 237–285.

[15] Jens Kober and Jan Peters. 2011. Policy Search for Motor Primitives in Robotics. *Machine Learning* 84, 1-2 (2011), 171–203.

[16] Rogier Koppejan and Shimon Whiteson. 2011. Neuroevolutionary reinforcement learning for generalized control of simulated helicopters. *Evolutionary Intelligence* 4, 4 (01 Dec 2011), 219–241.

[17] Manon Legrand. 2017. *Deep Reinforcement Learning for Autonomous Vehicle Control among Human Drivers*. Ph.D. Dissertation. Universite Libre de Bruxelles.

[18] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end Training of Deep Visuomotor Policies. *Journal Machine Learning Research* 17, 1 (Jan. 2016), 1334–1373.

[19] J.A. Martin H and J. de Lope. 2009. Ex<a>: An effective algorithm for continuous actions Reinforcement Learning problems. In *Industrial Electronics, IECON '09. 35th Annual Conference of IEEE*. IEEE, 2063 –2068.

[20] José Antonio Martín H. and Javier de Lope. 2009. Learning Autonomous Helicopter Flight with Evolutionary Reinforcement Learning. In *Computer Aided Systems Theory - EUROCAST 2009*, Roberto Moreno-Díaz, Franz Pichler, and Alexis Quesada-Arencibia (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 75–82.

[21] Risto Miikkulainen. 2017. Evolution of neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 450–470.

[22] Jean-Baptiste Mouret and Konstantinos I. Chatzilygeroudis. 2017. 20 years of reality gap: a few thoughts about simulators in evolutionary robotics. In *Genetic and Evolutionary Computation Conference, Berlin, Germany, July 15-19, 2017, Companion Material Proceedings*. 1121–1124.

[23] Andrew Ng, Jin Kim, Michael Jordan, and Shankar Sastry. 2003. Autonomous Helicopter Flight via Reinforcement Learning. In *NIPS*, S. Thrun, L. K. Saul, and B. Schölkopf (Eds.). MIT Press, 799–806.

[24] Jan Peters and Stefan Schaal. 2008. Reinforcement Learning of Motor Skills with Policy Gradients. *Neural Networks* 21, 4 (May 2008), 682–697.

[25] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Geoffrey Gordon, David Dunson, and Miroslav DudÃŋk (Eds.), Vol. 15. PMLR, Fort Lauderdale, FL, USA, 627–635.

[26] Juan C. Santamaría, Richard S. Sutton, and Ashwin Ram. 1998. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior* 6 (1998), 163–218. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.8280

[27] Bradly C. Stadie, Sergey Levine, and Pieter Abbeel. 2015. Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models. In *Neural Information Processing Systems*.

[28] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA.

[29] Lei Tai, Giuseppe Paolo, and Ming Liu. 2017. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *Intelligent Robots and Systems (IROS)*. IEEE, 31–36.

[30] Matthew E. Taylor, Brian Kulis, and Fei Sha. Metric Learning for Reinforcement Learning Agents. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 777–784.

[31] Matthew E. Taylor, Halit Bener Suay, and Sonia Chernova. 2011. Integrating Reinforcement Learning with Human Demonstrations of Varying Ability. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2 (AAMAS '11)*. International Foundation for Autonomous Agents and Multiagent Systems, 617–624.

[32] Lisa Torrey and Matthew E. Taylor. 2012. Help an Agent Out: Student/Teacher Learning in Sequential Decision Tasks. In *Proceedings of the Adaptive and Learning Agents workshop (at AAMAS-12)*. International Foundation for Autonomous Agents and Multiagent Systems, 41–48.

[33] Haddo van Hasselt and Marco Wiering. 2007. Reinforcement Learning in Continuous Action Spaces. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. IEEE, 272–279.