Policy Improvement through Safe Reinforcement Learning in High-Risk Tasks

Javier García
Computer Science Department
Universidad Carlos III de Madrid
Avenida de la Universidad 30
28911 Leganés, Madrid, Spain
fjgpolo@inf.uc3m.es

Fernando Fernández
Computer Science Department
Universidad Carlos III de Madrid
Avenida de la Universidad 30
28911 Leganés, Madrid, Spain
ffernand@inf.uc3m.es

***Abstract*—Reinforcement Learning (RL) methods are widely used for dynamic control tasks. In many cases, these are high risk tasks where the trial and error process may select actions which execution from unsafe states can be catastrophic. In addition, many of these tasks have continuous state and action spaces, making the learning problem harder and unapproachable with conventional RL algorithms. So, when the agent begins to interact with a risky and large state-action spaces task, an important question arises: how can we avoid that the exploration of the state-action space cause damages in the learning (or other) systems. In this paper, we define the concept of *risk* and address the problem of safe exploration in the context of RL. Our notion of safety is concerned with states that can lead to damage. Moreover, we introduce an algorithm that safely improve sub-optimal but robust behaviors for continuous state and action control tasks, and that learns eficiently from the experience gathered from the environment. We report experimental results using the helicopter hovering task from the RL Competition.**

I. Introduction

When using Reinforcement Learning (RL) techniques for learning in high-risk control tasks, traditionally the exploration/exploitation process (e.g. $\epsilon - greedy$) can lead to states or actions that can be catastrophic for the agent which is learning [10]. The helicopter hovering control task is an example of these tasks. In this control task, the agent's objective is to hover the helicopter by manipulating different continuous control inputs based on a continuous high-dimensional state space. This domain involves high risk, since some policies can crash the helicopter, incurring in catastrophic negative reward. Exploration/exploitation strategies such as $\epsilon - greedy$ may produce the helicopter constantly crash (especially if there is a high probability of choosing actions randomly). As result of this, the exploration process, in which new policies are evaluated, must be conducted with extreme care. For those environments, a method that does not only explore the state-action space, but that it explores safely, is required.

In this paper we propose a method for safe learning in high-risk and continuous control tasks. The method requires a previously defined safe policy, which we assume to be sub-optimal (otherwise, learning should have no sense). The goal is to learn a near-optimal policy, although optimality can not be warranted. We report experimental results of the new algorithms to learn policies for the helicopter hover task from the RL Competition, where we propose to learn a near-optimal policy minimizing the helicopter crashes during the learning phase.

Thus, we introduce *Policy Improvement through Safe Reinforcement Learning* (PI-SRL) as a novel approach for improving baseline policies using RL in high-risk domains. The PI-SRL algorithm is composed by two different steps. In the first one, a baseline behavior (robust but sub-optimal) is approximated by learning. Baseline behaviors can be modeled using Learning from Demonstration (LfD) techniques [13]. Within LfD, a teacher provides example executions of the task. Executions are recorded as observations of the world and selected actions. From this dataset, the learner then generalizes a control policy which maps observations to actions. However, some works demonstrated the ability to improve a policy based on the experience from a teacher [3], [4]. In the case of the helicopter hover task, the competition software provided a baseline controller. This baseline controller is robust, meaning that it never causes the helicopter to crash. However, its performance is quite weak, as it is unable to consistently hover near the target point. The baseline controller is a linear function: a direct mapping of an observation to an action. So, in the first step of the PI-SRL algorithm, we use this baseline controller as a teacher to build a simple behavior to complete the task. In order to achieve this goal, we use Case-Based Reasoning (CBR) techniques [1], which have been applied successfully for LfD in the past [8], [9].

In the second step of the PI-SRL algorithm, it tries to safely build a more accurate policy from the previous behavior learned from demonstration. Thus, the set of instances obtained in the previous phase is improved exploring the state-action space safely. To perform this exploration, small amounts of Gaussian noise are randomly added to the greedy actions of the base-line policy approximation. This exploration strategy has been used successfully in previous works [5], [22]. The novelty of this work is the use of two new main components: a *safety function* to determine a state's degree of safety and a *baseline policy behavior* that is able to lead the system from a critical state back to a safe one. Using these two components, the problem of safe exploration is basically solved.

The remainder of the paper is organized as follows. Next

section introduces the learning approach proposed, while Section III shows the evaluation performed in the helicopter hover domain. Section IV reports the related work and last, Section V summarizes the main conclusions of this work.

II. The PI-SRL Algorithm

To illustrate the concept behind this approach, consider the navigation problem shown in Figure 1. There, the task is to learn a control policy to get from the start state to the goal state, given a set of demonstration trajectories.

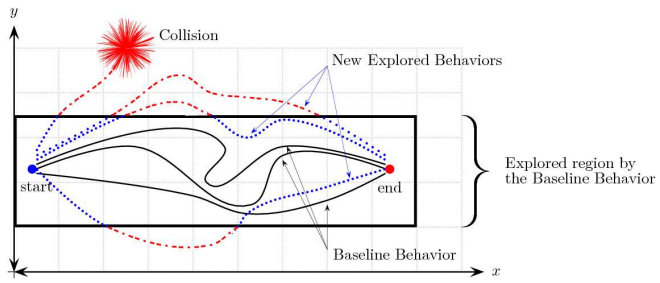

Fig. 1. Exploration strategy based on adding small amounts of noise to a baseline policy behavior. The baseline behavior is shown by filled lines and the new explored behaviors are shown by dotted lines.

In this environment, we assume that the task is hard due to a atochastic and complex dynamic of the environment (e.g. an extremely irregular surface in the case of the robot navigation domain, wind effects in the case of the helicopter hover task, ...). That makes impossible to complete the task using exactly the same trajectory every time. Now consider that we are given a set of demonstrations of a baseline controller performing the task (shown as filled black lines in Figure 1). This set of demonstrations is composed by different trajectories, covering a well-defined region of the state space (the region within the rectangle in Figure 1).

Our approach is based on adding small amounts of noise or perturbations to the baseline trajectories in order to find new and better ways to complete the task. This noise will affect the baseline trajectories seen in different ways, depending on the amount of noise added (or depending on the amount of *risk* you want to take). If you do not want to take *risks*, the noise added to the baseline trajectories should be 0, and no new nor better behaviors will be discovered (but the robot will never fall off a cliff, or the helicopter will never collide). If you want to take an intermediate level of *risk*, small amounts of noise can be added to the baseline trajectories, and new trajectories to complete the task are discovered (the blue dotted lines in Figure 1). In some cases, the exploration of new trajectories, leads the robot to unknown regions of the state space (the red dashed lines in Figure 1). The robot should be able to detect these situations (*safety function*), and to use the *baseline policy behavior* to return to safe and known states. If, instead, you want to take a very high *risk*, big amounts of perturbations will be added to the baseline trajectories, and new trajectories will be discovered (but high *risk* increases the probability of damaging the robot).

The iteration of this process leads the robot to explore progressively in a safe way the state and action spaces, in order to find new and better ways to complete the task. Safety will depend on the risk taken.

Next we describe the first step of the PI-SRL algorithm, where the agent learns (build a case base) by observing the behavior of a teacher. Later, the second step of the PI-SRL algorithm will be explained. In this step, the PI-SRL algorithm builds a more complex policy from the simpler policy previously learned from demonstration.

A. First Step: Modeling the Base-line Behaviors by CBR

In this section, we describe an approach for learning by observation that uses CBR to allow a software agent to behave similarly to a teacher. LfD techniques attempt to shift the burden of knowledge transfer from the teacher to the software agent. The agent learns by watching the teacher perform a task and, when faced with the same task, aims to behave in a similar manner [9].

When using CBR for behavior imitation, a case can be built using the agent's visual information as well as the corresponding action command it perform. A first approach to imitate the teacher behavior would be store all cases generated by the teacher. This idea seems intuitive, however, storing all observing cases is not useful when trying to imitate the teacher behavior. To illustrate this, consider Figure 2. In addition, consider the teacher behavior is described by the function $f(s) = a$ (i.e. given the observation/state *s*, the teacher perform the action *a*), and, if a new case is presented, a 1-nearest-neighbor search is used in order to find the best case match. The Figure 2 shows the region of the space represented by simple storing cases in the form (s_k, a_k) performed using f. Each stored case covers an area of the space (i.e. each stored instance, red circles in the figure, represents the centroid of a Voronoi region).

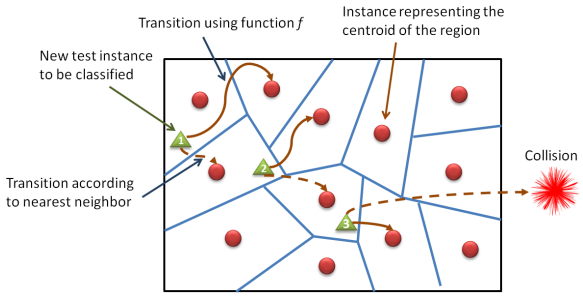

Fig. 2. Effects of storing all training instances.

When a new test case s_k is given (green triangles in Figure 2), the algorithm search for the most similar stored case, and classifies the new case according to the class a_k of its nearest neighbor. This classification results in the execution of the action a_k, which will produce an state transition, and a new input case (an state) s_{k+1} to be classified.

However, if the function f is used to classify the case s_k, probably the output would be slightly different to a_k,

and, therefore, the new input case to classify is also different to s_{k+1}. These classification errors lead to visit unexplored regions of the case space, not considered in the initial set of cases (the set of cases obtained by simple storing tuples in the form (s_k, a_k) of the function f). This is particularly dangerous in high-risk domains. When visiting an unexplored region, a big difference between the real output (obtained by the function f), and the estimated output (obtained by 1-nearest-neighbor classification), may cause damage in the robot (e.g. the helicopter crash), and our intention to imitate the teacher behavior f by CBR would fail.

So, simply storing training cases is not enough to imitate the teacher behavior by CBR in many domains. To model a base-line behavior, the algorithm depicted in Figure 3 has been developed. 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 *n*-dimensional action space $A \subset \Re^n$ 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 $D_i^a \subset \Re$. Accordingly, we define a case c to be an $n + m$-dimensional real-valuated vector $c = (s_1, \ldots, s_n, a_1, \ldots, a_m)$, where the first n elements represent the case's problem part and corresponds to state s, and the last m elements depicts the case's solution, i.e. the expected action when the agent is in the state s.

In this algorithm, the cases of the case base *CB* (representing the teacher behavior f) are evaluated in each step of an episode. The parameter θ is a threshold used to identify if a new state s_k belongs to an unexplored region of the state space. If so, the algorithm perform the action a_k using the function $f(s_k)$, and a new instance, $c^{new} = (s_k, a_k)$ is built and added to the case base CB. Otherwise, a 1-NN strategy is followed. Later, it calls the appropriate case base management routine, which is described bellow.

CBR Approach for Teacher Imitation

Given the function $f(s) = a$ describing the teacher behavior
1. Set the case base $CB = \emptyset$
2. **Repeat**
  Set $k = 0$
  while $k < maxEpisodeLength$ **do**
    Compute the instance $< s_c, a_c >$ closest to the current state s_k
    if $distance(s_c, s_k) < \theta$
      Set $a_k = a_c$
    else
      Set $a_k = f(s_k)$
      create a new case $c^{new} = (s_k, a_k)$
      $CB := CB \cup c^{new}$
      call case base management routines
    Execute a_k, and receive s_{k+1}
    Set $k = k + 1$
  end while
until stop criterion becomes true

Fig. 3. CBR algorithm for approximation of functions describing policy behaviors.

In the algorithm shown in Figure 3, the distance between states is computed using the Euclidean distance (Equation 1).

$$distance(s, s') = \sqrt{\sum_{i=0}^{n} (s_i - s'_i)^2} \quad (1)$$

Starting with an empty case base, the learning algorithm continuously increases its competence by storing new experiences. However, there are a number of reasons why the inflow of new cases ought to be limited. Large case bases increases the time required to find the closest instances of a new instance presented. This can be partially solved by using techniques to reduce the retrieval time, e.g. *kd-trees*, that have been used in this work. However, they does not reduce the storage requirements. Several approaches to remove *useless* cases during training exist, e.g. the IBx algorithms by Aha [2] or any nearest prototype approach. Anyway, when the number of cases stored in *CB* exceeds some critical value $\|CB\| > \mu$, so that the realization of a retrieval within a certain amount of time cannot be guaranteed, it is inevitable to remove some cases. An efficient approach to tackle that problem is to remove the least frequently used elements of *CB* (Figure 4).

Case Base Management Routine

1. Given the set of instances CB
2. **if** $\|CB\| \leq caseBaseMaxSize$ **return**
3. **for all** $c^i \in CB$
    Compute $t(c^i)$ with $t(c^i)$ telling how many episodes ago c^i has been visited
4. Delete instances with $t(c) > \varphi$

Fig. 4. Case Base Management: Deletion of Stored Cases.

The result of this step is a constrained case base *CB* that mimics the teacher behavior described by the function $f(s) = a$.

B. Second Step: Improving the Baseline Behavior

In the second step of the PI-SRL algorithm, the baseline behavior learned in the previous step is improved by safe exploration of the state-action space. In this step, in order to evaluate the goodness of an action in a particular state, a new field is added to each case $c \in CB$. Thus, each case $c \in CB$ is composed by (s, a, v), where s represents the case's problem part (or state), a depicts the case's solution (or action), and the utility value v represents an evaluation of how good the action a is for the state s. To initialize the utility value v for each state s, the state-value function is computed, $V^\pi(s)$, following a Monte Carlo (MC) approach (Figure 5).

This algorithm is similar in spirit to First-visit MC method for V^π [21]. In algorithm shown in Figure 5, all the returns for each state $s \in CB$ are accumulated and averaged, following the policy π defined by the instances in *CB*.

Once the utility value v is computed for each instance $c \in CB$, small amounts of Gaussian noise are randomly

MC Algorithm

1. Given the case base CB
2. Initialize, **for each** $c^i \in CB$
   $v^i \leftarrow$ *arbitrary*
   $\pi(s) \leftarrow a$ given by the instance $c^i =< s, a, v > \in CB$
   $Returns(s) \leftarrow$ *empty list*
3. **while** $k < maxNumberEpisodes$
   Generate an episode using π
   for each s appearing in the episode with $< s, a, v > \in CB$
   $R \leftarrow$ return following the first occurrence of s
   Append *R* to *Returns(s)*
   $v \leftarrow$ *average(Returns(s))*
4. **Return** CB

Fig. 5. Monte Carlo Algorithm to Compute the State-Value Function for each Instance.

added to the actions of the policy in *CB*, in order to obtain new and better ways to complete the task. Thus, the Gaussian exploration take place around the current approximation of the optimal action. The action that is performed is sampled from a Gaussian distribution with the mean at the action output given by the selected instance in *CB*. When a_t denotes the action that the algorithm outputs at time *t*, the probability of selecting action a'_t is computed using Equation 2.

$$\pi(s_t, a'_t) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(a'_t - a_t)^2/2\sigma^2} \quad (2)$$

The algorithm used to improve the baseline behavior learned in the previous step is depicted in Figure 6. This algorithm is composed by four different steps performed in each episode.

- *Initialization step*. It corresponds with the step labeled as (a) in Figure 6. It initializes the list used to store the cases occurred during an episode, and it sets to 0 the accumulated reward counter of the episode.

- *Case Generation*. This step corresponds with the step labeled as (b) in Figure 6. In this step, the algorithm builds a case for each step in a episode. For each new state s_k the closest case $< s, a, v >$ is computed using the Euclidean distance (Equation 1). In order to determine the degree of safety of the new state s_k perceived, a *safety function* is used. This *safety function* computes the distance between the state s_k and the closest state *s*, $distance(s_k, s)$. If $distance(s_k, s) < \theta$ the state is considered a safe state (i.e. this state is less likely to damage the robot, and the robot can perform the *risky* action), because the state s_k is considered similar to the state *s* (i.e. the state s_k is considered within the explored region defined by the case base *CB*). The action a_k performed in this case is computed using the Equation 2, and a new case $< s, a_k, v >$ is built to be added to the list of cases occurred during the episode.

If instead $distance(s_k, s) > \theta$ the state s_k is considered a dangerous state (i.e. in this state the robot is likely to be damaged if the action suggested by the *CB* is performed), because the state s_k is considered outside the region defined by the case base *CB*. In this dangerous state, a *baseline policy behavior* is needed to lead the system back to a safe state. So, in dangerous states, the action a_k performed is suggested by the teacher function f, which define a safe behavior. A new case $< s_k, a_k, 0 >$ is built to be added to the list of cases in the episode. Until the robot is not in a considered safe state, actions will be performed using the teacher behavior.

Finally, in this step of the algorithm, the total reward obtained in the episode is accumulated ($totalRwEpisode := totalRwEpisode + r(s_k, a_k)$, where $r(s_k, a_k)$ is the immediate reward obtained when perform action a_k in state s_k).

- *Computing the state-value function for the dangerous states*. This step corresponds with the step labeled as (c) in Figure 6. In this step, the state-value function of the states considered as dangerous states in the previous step are initialized (in the previous step the state-value function for these states was set to 0). The algorithm proceeds in a similar way than in Figure 5. In this case, all the returns for each dangerous state s_i are accumulated but not averaged, since only one episode is considered.

- *Updating the cases in CB using the experience gathered*. This step corresponds with the step labeled as (d) in the algorithm shown in Figure 6.

The update of the cases stored in *CB* only is performed when the accumulated reward obtained in the episode, *totalRwEpisode*, is greater than the greatest accumulated reward per episode obtained, *maxTotalRwEpisode*, minus a threshold Θ. Thus, the updates in *CB* will be made with the cases gathered from episodes with a similar quality to the best episode found so far.

In this step, there are two types of updates: replacements and additions of new cases. For each case $c_i = (s_i, a_i, V(s_i)) \in listCasesEpisode$, we compute if the problem part of the case c_i, s_i, is part of a case $u \in CB$. If so, then the TD-error, δ, is computed. If $\delta > 0$, performing the action a_i results in a positive change for the value of a state, then that action could potentially lead to a higher discounted future reward and thus to a better policy [22]. Therefore, we reinforce that action replacing the old case u by the new case c_i updating the state-value function of s_i, $V(s_i)$.

If instead, s_i is not part of a case in *CB*, the case c_i is added to *CB*, and the algorithm calls to the case base management routine (Figure 4).

III. Experimental Results

In this section, we report experimental results of using PI-SRL to learn policies in the well-known generalized helicopter hovering domain from the RL Competition [18]. In this domain, a RL agent seeks a policy for a simulated XCell Tempest helicopter. The goal is to make the helicopter hover as close as possible to a defined position for the duration established of an episode.

This domain represents a challenging task for four main reasons. First, transition dynamics are complex (i.e. how the state of the agents environment changes in response to its actions). Second, both the state and action spaces are high-dimensional and continuous (the state space is 12-dimensional and the action space 4-dimensional). Third, the helicopter

Policy Improvement Algorithm

Given the case base *CB*
Given the function $f(s) = a$ describing the teacher behavior
1. Set $maxTotalRwEpisode = 0$, the maximum cumulative reward reached in a episode
2. **Repeat**
   (a) **set** $k = 0$, *listCasesEpisode* $\leftarrow \emptyset$, $totalRwEpisode = 0$

   (b) **while** $k < maxEpisodeLength$ **do**
   Compute the case $< s, a, V(s_k) >$ closest to the current state s_k
   if $distance(s, s_k) < \theta$ **then**
   Perform action a_k using equation 2
   Create a new instance $c^{new} := (s, a_k, V(s_k))$
   else
   Perform action a_k, using $f(s_k) = a_k$
   Create a new instance $c^{new} := (s_k, a_k, 0)$
   $totalRwEpisode := totalRwEpisode + r(s_k, a_k)$
   $listCasesEpisode := listCasesEpisode \cup c^{new}$

   (c) **for each** instance c_i in $listCasesEpisode$
   if s_i is not part of a case in CB **then**
   $return(s_i) := \sum_{j=k}^{n-1} r(s_j, a_j) + r_{s_n}$
   $V(s_i) := return(s_i)$

   (d) **if** $totalRwEpisode > (maxTotalRwEpisode - \Theta)$ **then**
   if $totalRwEpisode > maxTotalRwEpisode$ **then**
   $maxTotalRwEpisode := totalRwEpisode$
   for each case c_i in $listCasesEpisode$
   if s_i is part of a case in CB **then**
   Compute the case $u \in CB$ corresponding to the state s_i
   Compute $\delta = r(s_i, a_i) + \gamma V(s_{i+1}) - V(s_i)$
   If $\delta > 0$ **then**
   Replace the case $u \in CB$ with c_i
   $V(s_i) = V(s_i) + \alpha\delta$
   else
   $CB := CB \cup c^i$
   call case base management routines

   until stop criterion becomes true
4. **Return** CB

Fig. 6. Description of the second step of the PI-SRL algorithm which perform the policy improvement.

hovering domain involve high risk, as bad policies could crash the helicopter, obtaining a catastrophic negative reward. Fourth, it is a generalized domain where wind factor modify the domain behavior. Since the 2008 RL Competition the wind in the helicopters environment varies greatly from one SDP to the next.

Now, we describe briefly the helicopter hovering problem. In each step, the helicopter receives a 12-dimensional continuous state and performs a 4-dimensional continuous action. The helicopter also receives a immediate reward that is the sum, over all state features, of the squared difference between that state feature and the fixed target position in which the helicopter wishes to hove. A helicopter episode is composed by 6000 steps, but an episode could end prematurely if the helicopter crashes. The helicopter's crashing results in a large negative reward. The goal is to find a control policy that maximizes the reward that the helicopter receives per episode. However, in this work, a new challenge is added to the default goal defined previously. In our case, the goal is to find the best possible control policy (as previously mentioned) but, in addition, minimizing the number of episodes where the helicopter crashes.

As mentioned before, the helicopter hovering domain provides use with a robust baseline controller (i.e. it never causes the helicopter crash). However, its performance is weak, and it is unable to hover the helicopter near of the specified target point. The baseline controller is a linear function, $f(s) = a$, which directly maps an state *s* perceived to an action *a*. The first step of the PI-SRL algorithm is performed in order to imitate this baseline controller. In this step, the teacher is considered as the baseline controller provided in the domain. Once the first step of PI-SRL is performed as described in section II-A, the resulting case base *CB* obtained, is able to properly imitate the robust behavior of the baseline controller (Figure 7).

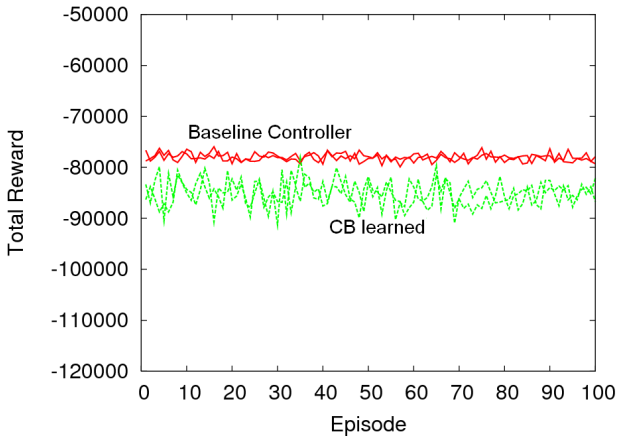

Fig. 7. Comparative of the performance of the Baseline Controller provided by the software competition, and the dataset *CB* learned once the first step of PI-SRL has been performed.

Figure 7 compares the performance (in terms of total reward per episode) for the baseline controller provided by the software competition, and the case base *CB* learned. The averaged total reward per episode of the Baseline Controller during 100 episodes is -78035.93, while the obtained for the case base *CB* learned is -85130.11. Although the case base *CB* do not mimic perfectly the behavior of the baseline controller, it performs a robust policy without crashing the helicopter.

Previous to perform the second step of the PI-SRL algorithm, the case base *CB* learned can be used to analyze some features of the helicopter domain: the reward function and the distribution of the actions.

To learn the reward function, a similar version of the algorithm described in Figure 5 can be used, but instead only the immediate reward for each state $s \in CB$ are accumulated and averaged, following the policy π defined by the instances in *CB*. The reward function learned is shown in Figure 8.

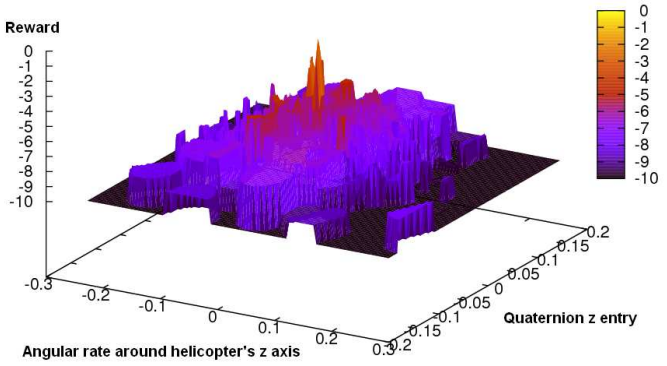

Fig. 8. Reward function of the helicopter domain.

Figure 8 shows how the bigger immediate rewards are around the origin of coordinates. So, to learn in the helicopter domain consists in to hover the helicopter as long as possible closest to the origin of coordinates, where a greater amount of reward will be collected.

The case base *CB* can also be used to study the spacial distributions of the actions (Figure 9). Figure 9 shows four different histograms, each one analyzing the distribution of one dimension of the actions in *CB* (i.e. each action is composed by four dimensions $a_1, a_2, \ldots, a_4$, so each histogram analyzes the distribution of an individual dimension in all the case base *CB*).

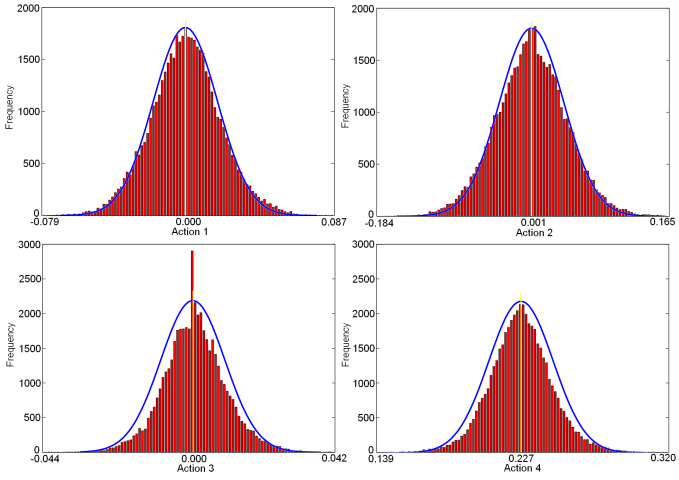

Fig. 9. Histograms analyzing the distribution of the individual dimension of actions (a histogram for each dimension) in all the case base *CB*.

Figure 9 shows how each dimension of actions in *CB* follows a gaussian distribution. This is particularly interesting, and reinforces the use of Gaussian exploration in the second step of the PI-SRL algorithm.

Once the case base *CB* is obtained in the first step of the PI-SRL algorithm, the second step is performed. In this step, PI-SRL tries to safely build a more complex policy from the previous behavior learned from demonstration. Thus, the set of instances *CB* obtained in the previous phase is improved exploring the state-action space safely. The results of the PI-SRL in this step, are compared with the results of the winner of the RL Competition 2009 in the helicopter hovering task [15]. The winner of the autonomous helicopter flight in 2009, used an evolutionary RL method in order to find a near optimal neuro-controller [17]. The comparative results are shown in Figure 10.

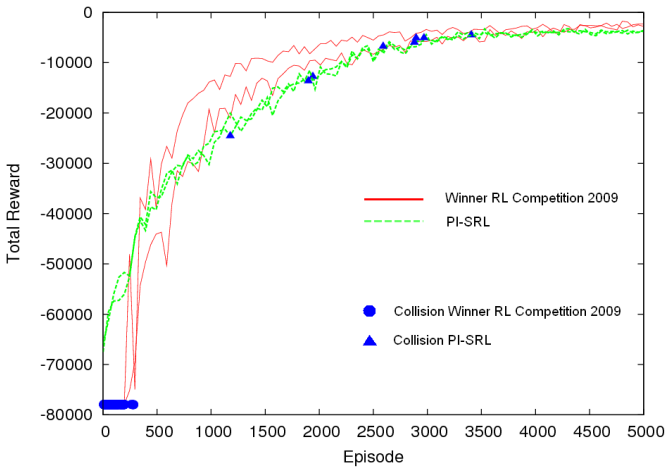

Fig. 10. Comparative learning results between the winner of RL Competition 2009 and the PI-SRL algorithm.

Figure 10 shows two learning curves of the winner of the RL Competition 2009 (red filled lines), and two learning curves of the PI-SRL algorithm (green dashed lines). In addition, it is marked when an episode ends with helicopter's collision (blue circles for the winner of RL Competition 2009 and blue triangles for the PI-SRL algorithm). The results show that we can obtain similar results with both methods, but the number of collisions when using PI-SRL (a mean of 4 collisions) is much lower than when using the evolutionary RL approach (a mean of 161 collisions). However, when using the evolutionary approach, all crashes occur in early steps of the learning process, while when using the PI-SRL algorithm, collisions occur in more advanced steps of the learning process.

Figure 11 analyzes the new distribution of the actions in the case base *CB* once the second step of the PI-SRL algorithm is performed.

Figure 11 demonstrates how all the dimensions of the action space have been moved slightly to the left, with respect to the distribution of the base-line behavior. Due to the case base *CB* obtained after performing the second step of PI-SRL has more cases than the previous one, the frequencies showed in the histograms are higher.

IV. Related Work

Both *Learning for Demonstration* (LfD) and *Safe Reinforcement Learning* are widely described in the literature. Learning a task from scratch may require a prohibitively time consuming amount of exploration of the state-action space in order to find a good policy. In addition, learning without prior knowledge seems to be an approach that is not taken in human learning. Knowledge about how to approach a new task can

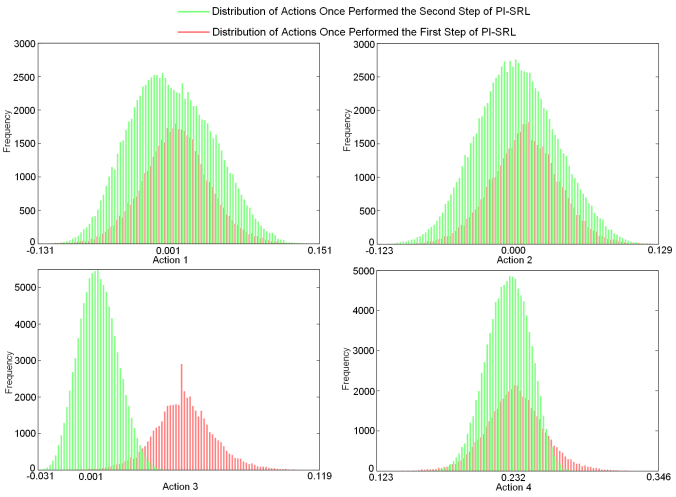

Fig. 11. Histograms analyzing the distribution of the individual dimension of actions (a histogram for each dimension) in all the case base *CB* (once the second step of the PI-SRL algorithm is performed).

be transferred from previously learned tasks, and/or it can be extracted from the performance of a teacher. This is the goal of LfD [20]. In the context of human skill learning, teaching by showing was investigated deeply in different works [14], [16], [7]. Although most LfD works to date has made use of human demonstrators, some techniques have also examined the use of robotic teachers [19], hand-written control policies or simulated planners []. A comprehensive survey of robot LfD techniques can be found in [5].

In the case of *Safe Reinforcement Learning*, the work has been mainly focused in determining when a state or transition can be considered as dangerous [12], [11], [10], and in the definition of *backup policies* able to lead the system to safe states from fatal states [12].

Other works combine successfully LfD with RL techniques. In [4], the learning system includes the ability to improve the policy based upon the learner experience. It derives new data from learner executions modified by advice from a human teacher. In [20] instead, LfD is combined with RL techniques in a different way to solve successfully the Swing-Up task and the Cart-Pole balancing. However, in theses works, the safe exploration of the state-action spaces is not mentioned, and probably the use of these techniques to learn in risky domains as the helicopter hovering control, would cause a big number of helicopter crashes.

In contrast with previous works, our approach is well-defined in two different steps. In the firs one, a baseline behavior is learned using LfD techniques. In the second step, the behavior learned previously is improved by safe exploration of the state-action spaces. The proposed approach is particularly suitable in risky domains, where to damage the robot result catastrophic.

V. Conclusion

In this work, PI-SRL, an algorithm for policy improvement through safe reinforcement learning in high-risk task, is described. We have studied the details of the algorithm and showed its effectiveness on the helicopter hovering task. The algorithm requires a previously defined safe policy, which is assumed to be sub-optimal. In this work, we have considered a *robotic teacher* to learn a robust (but sub-optimal) policy. This *robotic teacher* is provided by the software competition of the helicopter hovering.

In the second step of the PI-SRL algorithm, the previous learned robust behavior is improved through safe reinforcement learning. In this step, there are two main components: a *safety function* and a *baseline behavior* able to return to safe states from dangerous situations. In this work, a *safety function* based on the distance Euclidean between the new state perceived and the state space known by the agent is used. The use of this *safety function* is possible because the behavior policy is stored as a case base. This represents a clear advantage compared to other approaches (e.g. evolutionary reinforcement learning, where extraction of knowledge about the state space known by the agent is impossible using the weights of the neural-networks). In addition, the *robotic teacher* used in the previous step, is used again in this one. It represents the *baseline behavior*, used when the agent is in considered dangerous situations.

The algorithm works well, and achieves a performance that is competitive with the best published results. In addition, the algorithm is able to learn a near-optimal policy minimizing the helicopter crashes.

Several issues related to this work merit further research. In this work, a *robotic teacher* have been used. As future work, we attempt to use *human teacher* in other domains such as business administration [6]. In this domain the agent manages a company and it tries to get the maximum profits as possible in a competitive market. In this case, the *human teacher* would be an businessman expert, and the risk is to explore the state-action spaces without breaking the company. Finally, the exploration/explotaition problem is one important issue that we have not currently addressed at all in this work. A logical continuation of the work presented here would be to learn a value function model, to generate more appropriate exploratory actions.

Acknowledgment

This work has been partially supported by the Spanish MICIIN projects TIN2008-06701-C03-03 and TRA2009-0080

References

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

[2] David W. Aha. Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. *Int. J. Man-Mach. Stud.*, 36(2):267–287, 1992.

[3] Brenna Argall, Brett Browning, and Manuela Veloso. Learning mobile robot motion control from demonstrated primitives and human feedback. In *In Proceedings of the 14th International Symposium on Robotics Research (ISRR09)*, 2009.

[4] Brenna Argall, Brett Browning, and Maria Manuela Veloso. Learning robot motion control with demonstration and advice-operators. In *n Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2008.

[5] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, May 2009.

[6] Fernando Borrajo, Yolanda Bueno, Isidro de Pablo, Begoña Santos, Fernando Fernández, Javier García, and Ismael Sagredo. Simba: A simulator for business education and research. *Decis. Support Syst.*, 48(3):498–506, 2010.

[7] R. Dillmann, M. Kaiser, and A. Ude. Acquisition of elementary robot skills from human demonstration. In *In International Symposium on Intelligent Robotics Systems*, pages 185–192, 1995.

[8] Michael W. Floyd and Babak Esfandiari.

[9] Michael W. Floyd, Babak Esfandiari, and Kevin Lam. A case-based reasoning approach to imitating robocup players. In *Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference, May 15-17, 2008, Coconut Grove, Florida, USA*, pages 251–256, 2008.

[10] Peter Geibel. Reinforcement learning with bounded risk. In *In Proceedings of the Eighteenth International Conference on Machine Learning*, pages 162–169. Morgan Kaufmann, 2001.

[11] Peter Geibel and Fritz Wysotzki. Risk-sensitive reinforcement learning applied to chance constrained control. *JAIR*, 24, 2005.

[12] Hans, Alexander, Schneegass, Daniel, Anton M. Schäfer, and Udluft, Steffen. Safe Exploration for Reinforcement Learning. In *European Symposium on Artificial Neural Network*, pages 143–148, April 2008.

[13] J. A. Ijspeert, J. Nakanishi, and S. Schaal. movement imitation with nonlinear dynamical systems in humanoid robots. In *international conference on robotics and automation (icra2002)*, 2002.

[14] M. Kawato, F. Gandolfo, H. Gomi, and Y. Wada. Teaching by showing in kendama based on optimization principle. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN'94)*, 1994.

[15] José Antonio Martín H. and Javier Lope. Learning autonomous helicopter flight with evolutionary reinforcement learning. pages 75–82, 2009.

[16] Hiroyuki Miyamoto, Stefan Schaal, Francesca Gandolfo, Hiroaki Gomi, Yasuharu Koike, Rieko Osu, Eri Nakano, Yasuhiro Wada, and Mitsuo Kawato. A kendama learning robot based on bi-directional theory. *Neural Netw.*, 9(8):1281–1302, 1996.

[17] David Moriarty, Alan Schultz, and John Grefenstette. Reinforcement learning through evolutionary computation. Technical report, Journal on Artificial Intelligence Research (JAIR). submitted, 1999.

[18] Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. Autonomous helicopter flight via reinforcement learning. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors, *NIPS*. MIT Press, 2003.

[19] Monica N. Nicolescu and Maja J Mataric. Experience-based representation construction: Learning from human and robot teachers. In *In Proc., IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, pages 740–745, 2001.

[20] S. Schaal. learning from demonstration. In *advances in neural information processing systems 9*, pages 1040–1046. mit press, 1997.

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998.

[22] H. van Hasselt and M. A. Wiering. Reinforcement learning in continuous action spaces. In *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*, pages 272–279, 2007.