PIECEWISE CONSTANT REINFORCEMENT LEARNING FOR ROBOTIC APPLICATIONS

Andrea Bonarini, Alessandro Lazaric and Marcello Restelli
Department of Electronics and Information, Politecnico di Milano
piazza Leonardo Da Vinci, 32
20133, Milan, Italy

Keywords: Robot Learning, Reinforcement Learning.

Abstract: Writing good behaviors for mobile robots is a hard task that requires a lot of hand tuning and often fails to consider all the possible configurations that a robot may face. By using reinforcement learning techniques a robot can improve its performance through a direct interaction with the surrounding environment and adapt its behavior in response to some non-stationary events, thus achieving a higher degree of autonomy with respect to pre-programmed robots. In this paper, we propose a novel reinforcement learning approach that addresses the main issues of learning in real-world robotic applications: experience is expensive, explorative actions are risky, control policy must be robust, state space is continuous. Preliminary results performed on a real robot suggest that on-line reinforcement learning, matching some specific solutions, can be effective also in real-world physical environments.

1 INTRODUCTION

A lot of research efforts have been spent in robotics to identify control architectures with the aim of making the writing of control programs easier. Although many advances have been made, it is often difficult for a programmer to specify how to achieve the desired solution. Furthermore, it is hard to take into consideration all the possible configurations the robot may face or the changes that may occur in the environment.

Reinforcement Learning (RL) (Sutton and Barto, 1998) is a well-studied set of techniques that allow an agent to achieve, by trial-and-error, optimal policies (i.e., policies that maximize the expected sum of observed rewards) without any a priori information about the problem to be solved. In the RL paradigm, the programmer, instead of programming how the robot should behave, has just to specify a reward function that models how good is an action when taken in a given state. This level of abstraction allows to write specifications for the robot behavior in a short time and to obtain better and more robust policies with respect to hand-written control code.

Despite the huge research efforts in the RL field, the application of RL algorithms to real-world robotic problems is quite limited. The difficulty to gather experience, the necessity to avoid dangerous configurations, the presence of continuous state variables are some of the features that make the application of RL techniques to robotic tasks complex.

In this paper, we analyze the main issues that must be faced by learning robots and propose a set of techniques aimed at making the RL approach more effective in real robotic tasks. In particular, our main contributions are the introduction of the *lower bound* update strategy, which allows to learn robust policies without the need of a complete exploration of the whole state-action space, and the use of *piecewise-constant policies* with reward accumulation, which allows to efficiently learn even in presence of coarse discretizations of the state space. Experimental results carried out with a real robot show that the proposed learning techniques are effective in making the learning process more stable than traditional RL algorithms.

In the next section, we will briefly review the main approaches proposed in literature to overcome the problems described above. In Section 3, we introduce the RL framework and present the details of our

Bonarini A., Lazaric A. and Restelli M. (2007).
PIECEWISE CONSTANT REINFORCEMENT LEARNING FOR ROBOTIC APPLICATIONS.
In *Proceedings of the Fourth International Conference on Informatics in Control, Automation and Robotics*, pages 214-221
DOI: 10.5220/0001649102140221

algorithm. The results of the experimental validation, carried out on a real robot with a soccer task, are reported in section 4. We draw conclusions and propose future research directions in section 5.

2 REINFORCEMENT LEARNING ON ROBOTS

Given the complexity of the development of robotic applications, the possibility to exploit learning techniques is really appealing. In particular, the reinforcement learning research field provided a number of algorithms that allow an agent to learn to behave optimally by direct interaction with the environment without any a priori information.

Up to now, excluding a few notable exceptions, the application of the RL approach has been successful only in small gridworlds and simple simulated control problems. The success obtained in more complex domains (Tesauro, 1995; Sutton, 1996) is mainly due to ad-hoc solutions and the exploitation of domain dependent information.

Several difficulties prevent the use of pure RL methods in real world robotic applications. Since direct experience is the main source of information for an RL algorithm, the learning agent needs to repeatedly interact with the world executing each available action in every state. While in software domains it is possible to perform a large number of trials and to place the agent in arbitrary states, in real robotic domains there are several factors (limited battery capacity, blocking states, mechanical or electrical faults, etc.) that make learning from scratch not feasible.

Several works in the robotic area have studied different solutions to reduce the amount of direct experience required by RL algorithms. A typical solution consists of performing extensive training sessions using a physical simulator (Morimoto and Doya, 2000). When the simulated robot achieves a good performance the learned policy is applied on the physical robot and the learning process goes on with the aim of adjusting it to the real conditions. Although this approach can be really effective, for many robotic domains it is too hard to realize good enough simulation environments. Furthermore, it may happen that the approximation introduced in the simulation is such that the knowledge gathered in the simulated learning phase is almost useless for the real robot. Another approach that was originally proposed for robotic tasks, but that has found common application in other domains, is experience replay (Lin, 1992). The idea is that the robot stores data about states, actions, and rewards experienced, and fictitiously repeats the same moves thus performing more updates that speed up the propagation of the rewards and the convergence to the optimal value function. Although this approach succeeds in speeding up the learning process, it still requires an expensive exploration phase to gather enough information. To overcome this problem, Lin adopts a human teacher to show the robot several instances of reactive sequences that achieve the task in order to bias the exploration to promising regions of the action space. The same goal is pursued in (Millán, 1996), but instead of a teacher it requires a set of pre-programmed behaviors to focus the exploration on promising parts of the action space when the robot faces new situations. In this paper, we follow the approach proposed by (Smart and Kaelbling, 2002), which effectively provides prior knowledge by splitting the learning process in two phases. In the first phase, example trajectories are supplied to the robot (by automatic control or by human guidance) through a control policy and the RL system passively watches the experienced states, actions and rewards with the aim of bootstrapping information into the value-function. Once enough data has been collected, the second phase starts and the robot is completely controlled by the RL algorithm. In problems with sparse reward functions, without any hint, the robot would take a huge number of steps before collecting some significant reward, thus making the learning process prohibitive. At the opposite, the "supervised" phase is an initialization of the learning process so that it can initially avoid a fully random exploration of the environment. Furthermore, differently from imitation learning methods, this approach allows to supply prior knowledge without knowing anything about inverse kinematics.

Learning in real-world environments requires to deal with dangerous actions that may harm the robot or humans, and with stalling states, i.e., configurations that prevent the robot from autonomously going on with the learning process. Again, example trajectories can be effective to provide safe policies that avoid harmful situations. Another way to reduce the risk of performing dangerous actions is to use minimax learning (like the $\widehat{Q}$-learning algorithm (Heger, 1994)), where the robot, instead of maximizing the expected sum of discounted rewards, tries to maximize the value of the worst case. This kind of pessimistic learning has lead to good results in stochastic (Heger, 1994), partially observable (Buffet and Aberdeen, 2006), and multi-agent (Littman, 1994) problems, showing also to be robust with respect to changes in the problem parametrization, thus allowing the reuse of the learned policy in different operating conditions. Unfortunately, algorithms like $\widehat{Q}$-

learning need to perform an exhaustive search through the action space, but this is not feasible in a real-world robotic context. In the following sections, we propose a variant to the $\widehat{Q}$-learning that is able to find safe policies without requiring the complete exploration of the action space.

Another relevant issue in real-world robotic applications is that both the state and action spaces are continuous. Usually, this problem is faced by using function approximators such as state aggregation, CMAC (Sutton, 1996) or neural networks, in order to approximate the value function over the state space. Although these techniques obtained relevant results in supervised learning, they require long hand tuning and may result in highly unstable learning processes and even divergence. Although state aggregation is one of the most stable function approximator, its performance of state aggregation is strictly related to the width of the aggregated states and algorithms like Q-learning may have very poor performance even in simple continuous problems, unless a very fine discretization of the state space is used. In this paper, we propose a learning technique that allows to achieve good policies even in presence of large state aggregations, thus exploiting their generalization properties to reduce the learning times.

3 THE ALGORITHM

As discussed in previous sections, learning in noisy continuous state spaces is a difficult task for many different reasons. In this section, after the introduction of the formal description of the RL framework, we detail a novel algorithm based on the idea of the computation of a lower bound for a piecewise constant policy.

3.1 The Reinforcement Learning Framework

RL algorithms deal with the problem of learning how to behave in order to maximize a reinforcement signal by a direct interaction with a stochastic environment. Usually, the environment is formalized as a finite state discrete Markov Decision Process (MDP):

1. A set of states $\mathcal{S} = \{s_1, s_2, \cdots s_N\}$
2. A set of actions $\mathcal{A} = \{a_1, a_2, \cdots a_M\}$
3. A transition model $\mathcal{P}(s, a, s')$ that gives the probability to get to state s' from state s by taking action a
4. A reward function $\mathcal{R}(s, a)$ that gives the value of taking action a in state s

Furthermore, each MDP satisfies the Markov property:

$$\mathcal{P}(s_{t+1}|s_t, a_t) = \mathcal{P}(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \cdots, s_0, a_0), \quad (1)$$

that is, the probability of getting in state s at time $t+1$ depends only on the state and action at the previous time step and not on the history of the system.

A deterministic policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is a function that maps each state in the environment to the action to be executed by the agent. The action value function $Q^\pi(s)$ measures the utility of taking action a in state s and following a policy π thereafter:

$$Q^\pi(s,a) = R(s,a) + E\left[\sum_{t=1}^{\infty} \gamma^t R(s(t), \pi(s(t)))\right], \quad (2)$$

where $\gamma \in [0,1)$ is a discount factor that weights recent rewards more than those in the future.

The goal of the agent is to learn the optimal policy π^* that maximizes the expected discounted reward in each state. The action value function corresponding to the optimal policy can be computed by solving the following Bellman equation (Bellman, 1957):

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} \mathcal{P}(s,a,s') \max_{a'} Q^*(s',a'). \quad (3)$$

Thus, the optimal policy can be defined as the greedy action in each state:

$$\pi^*(s) = arg\max_a Q^*(s,a). \quad (4)$$

Q-learning (Watkins and Dayan, 1992) is a model-free algorithm that incrementally approximates the solution through a direct interaction with the environment. At each time step, the action value function $Q(s,a)$ is updated according to the reward received by the agent and to the estimation of the future expected reward:

$$Q^{k+1}(s,a) = (1-\alpha)Q^k(s,a) + \alpha\left[r + \gamma \max_{a'} Q^k(s',a')\right],$$

where $r = R(s,a)$ and α is the learning rate. In the following, we will refer to the term in square brackets as the *target value*:

$$U(s,a,s') = R(s,a) + \gamma \max_{a'} Q^k(s',a') \quad (5)$$

When α decreases to 0 according to the Robbins-Monro (Sutton and Barto, 1998) conditions and each state-action pair is visited infinitely often, the algorithm is proved to converge to the optimal action value function. Usually, at each time step the action to be executed is chosen according to an explorative policy that balances a wide exploration of the environment and the exploitation of the learned policy. One of the most used exploration policies is ε-greedy (Sutton, 1996) that chooses the greedy action with probability $1-\varepsilon$ and a random action with probability ε.

3.2 Local Exploration Strategy

In order to avoid an exhaustive exploration of the action space, a more sophisticated exploration policy can be adopted. As suggested in (Smart and Kaelbling, 2002), a "supervised" phase performed using a sub-optimal controller is an effective way to initialize the action value function. This way, the learning process is bootstrapped by a hand-coded controller whose policy is optimized when the control of the robot is passed to the learning algorithm. Even if this technique is effective to avoid an initial random exploration, the ε-greedy exploration policy used thereafter does not guarantee that many useless, and potentially dangerous, actions are explored. To reduce this problem, we propose to adopt a *local* ε-greedy exploration policy. Since the learning process is initialized using a sub-optimal controller, it is preferable to perform an exploration in the range of the greedy action, instead of completely random actions. Therefore, with probability ε a locally explorative action is drawn from a uniform probability distribution over a δ-interval of the greedy action a^* [1]:

$$a_{EXP} \sim U(a^* - \delta; a^* + \delta) \tag{6}$$

Even if the *local* ε-greedy exploration policy is not guaranteed to avoid dangerous explorative actions, in many robotic applications it is likely to be safer and to converge to the optimal policy in less learning episodes than usual ε-greedy policy. In fact, it is based on the assumption (often verified in robotic applications) that the optimal policy can be obtained by small changes to the sub-optimal controller.

3.3 Q_{LB}-learning

As showed in many works (e.g., (Gaskett, 2003; Morimoto and Doya, 2001)) traditional Q-learning is often ill-suited for robotic applications characterized by noisy continuous environments with several uncertain parameters in which non-stationary transitions may occur because of external unpredictable factors. Many techniques (Gaskett, 2003; Morimoto and Doya, 2001; Heger, 1994) improve the stability and the robustness of the learning process, on the basis of the concept of maximization of performance in the worst case, i.e., the *min-max* principle. This principle deals with the problems introduced by highly uncertain and stochastic environments. In fact, the controller obtained at the end of the learning process is optimized for the worst condition and not for the average situation, such as in Q-learning. Although effective in principle, this approach cannot always be applied in real world applications. The *Robust Reinforcement Learning* (RRL) paradigm (Morimoto and Doya, 2001) relies on an estimation of the dynamics and of the noise of the environment in order to compute the min-max solution of the value function. Unfortunately, the model of the environment is not always available and the estimation of its dynamics often requires many learning episodes. A model-free solution, the $\widehat{Q}$-learning algorithm, proposed in (Heger, 1994) can learn a robust controller through a direct interaction with the environment. The update formula for the action value function is:

$$Q^{k+1}(s,a) = \min\left[Q^k(s,a), U(s,a,s')\right], \tag{7}$$

where $U(s,a,s')$ is the target value. If the action value function is initialized to the highest possible value (i.e., $Q(s,a) = \frac{R_{max}}{1-\gamma}$), this algorithm is proved to converge to the min-max value function and policy, that is the policy that receives the highest expected reward in the worst case. Although this algorithm is guaranteed to find a robust controller, it can be applied only to simulated environments, since the optimistic initialization of the action value function makes the agent to explore randomly all the available actions until at least the best action converged to the min-max value function. Thus, it is not suitable for robotic applications where long exploration is too expensive.

Another drawback of $\widehat{Q}$-learning is that it finds an optimal policy for the worst case even if caused by non-stationary transitions. In fact, in real-world applications very negative conditions may occur during the learning process because of very limited and uncontrolled situations possibly caused by non-stationarity in the environment and, with $\widehat{Q}$-learning, these conditions are immediately stored in the action value function and cannot be removed anymore.

In order to keep the robustness of a min-max controller, to reduce the exploration and to avoid effects of non-stationarity as much as possible, we propose Q_{LB}-learning, a novel algorithm for the computation of a lower bound for the action value function. Instead of a minimization between the current estimation and the target value (Eq. 5), we adopt the following update rule:

$$Q^{k+1}(s,a) = \begin{cases} U(s,a,s') & \text{if } U(s,a,s') < Q^k(s,a) \\ (1-\alpha)Q^k(s,a) + \alpha U(s,a,s') & \text{otherwise} \end{cases}$$

As it can be noticed, when the worst case is visited the action value estimation is set to the target value as in $\widehat{Q}$-learning. On the other hand, if the target received

[1] In case of problems with multiple actuators, the explorative action is obtained by the composition of explorative actions for each actuator.

by the agent is greater than the current estimation, the usual Q-learning update rule is used. As a result, the action value function may not take into consideration the worst case ever visited in the learning process, when this is the result of rare events not following the real dynamics of the system (e.g., collisions against moving obstacles). As learning progresses the learning rate α decreases (according to Robbins-Monro conditions) thus granting the convergence of the Q_{LB}-learning algorithm since it becomes more and more similar to $\widehat{Q}$-learning [2]. As it can be noticed, this algorithm does not require any particular initialization of the action value function as in $\widehat{Q}$-learning and this can reduce the exploration needed to learn a nearly optimal solution. Furthermore, this algorithm is effective in case of continuous state spaces in which the transitions between states may be affected also by the policy the robot is performing (Moore and Atkeson, 1995). In this situation, the worst case depends on the policy and not only on the dynamics of the environment, thus it is necessary to evaluate the action value function according to the current policy and not with respect to the worst possible case. Therefore, while $\widehat{Q}$-learning would converge to the worst case independently from the policy, Q_{LB}-learning learns the action value function for the worst case of the current policy.

3.4 PWC-Q-learning

Although the previous algorithm is effective in noisy or non-stationary environments, it may experience bad results (see Section 4) when applied to problems with continuous state spaces. Usually, when applying RL algorithms, a continuous state space is discretized into intervals that are considered as aggregated states. Unfortunately, if a coarse discretization is adopted, the environment looses the Markov property and the learning process is likely to fail. In fact, a very fine resolution on the state space is required to make algorithms such as Q-learning stable and effective. Since the number of states has a strong impact on the speed of the learning process, this is often a problem of its application to robotic problems.

The reason for Q-learning to fail in learning on coarsely discretized states is strictly related to its learning process, that continuously performs updates within the same state. With Q-learning, when the robot takes the greedy action a in state s, receives a reward r and remains in the very same state, the action value function is updated as:

$$Q^{k+1}(s,a) = (1-\alpha)Q^k(s,a) + \alpha\left[r + \gamma Q^k(s,a)\right] \quad (8)$$

[2] Let us notice that when $\alpha = 0$ Q_{LB}-learning is the same as the $\widehat{Q}$-learning

Thus, the value of $Q(s,a)$ is updated using its own estimation. If the state is sufficiently large, the agent is likely to remain in the same state for many steps and the Q-value tends to converge to its limit $\frac{r}{1-\gamma}$ until either the state is left or another action is chosen. As a result, the policy continuously changes and the learning process may experience instability. This phenomenon is much more relevant when a *min-max*-based update rule is adopted; in that case, the convergence to the limit is even faster.

In order to avoid the negative effect of the self-update rule, we introduce a novel learning algorithm: the *Piecewise Constant Q-learning* (PWC-Q-learning). The main difference with respect to the traditional Q-learning is about the way the action value function is updated. When the robot enters a state s and selects an action a, PWC-Q-learning makes the robot repeatedly execute the same action a and it accumulates the reward until a state transition occurs. Only at that time, the update is performed according to the SMDP Q-learning rule (Sutton et al., 1999):

$$\begin{aligned} Q^{k+1}(s,a) &= (1-\alpha)Q^k(s,a) + \\ &\quad \alpha\left[\sum_{i=1}^{N}\gamma^{i-1}r_i + \gamma^N \max_{a'} Q^k(s',a')\right] \end{aligned}$$

where N is the number of times in which the agent has performed action selection in the state s. As it can be noticed, in this way s' is always different from s and no self-update is performed. As a result, the PWC-Q-learning is more stable and guarantees a more reliable learning process than the original Q-learning.

Furthermore, the PWC-Q-learning algorithm matches also the limitations caused by the discretization on the resolution of the controller that can be actually learned. In fact, when the learning is over, the learned policy π maps each state into one single action that must be kept constant until a different state is perceived. Therefore, in this case a learning algorithm as PWC-Q-learning that evaluates the real utility of an action throughout a state is more suitable, and does not allow any change in the action as in Q-learning.

While Q-learning needs a very fine discretization to reduce the instability caused by the loss of the Markov property, PWC-Q-learning (as shown in the experimental section) proved to be more stable even in coarsely discretized continuous state spaces. By using coarse discretizations it is also possible to reduce the duration of the learning process.

Finally, PWC-Q-learning can be merged with the computation of the lower bound action value function introduced in Section 3.3 in order to obtain a learning algorithm that is robust in highly stochastic and noisy environments and that, at the same time, can be successfully applied to continuous robotic problems.

Figure 1: The RoboCup robot used for the experiments.

4 EXPERIMENTAL RESULTS

To verify the effectiveness of the proposed approach we made real robotic experiments with the aim of measuring speed and stability of the learning process, and optimality and robustness of the learned policy. The experimental activity has been performed on a real robot belonging to the Milan RoboCup Team (MRT) (Bonarini et al., 2006), a team of soccer robots that participates to the Middle Size League of the RoboCup competition (Kitano et al., 1997). The robot used for the experiments is a holonomic robot with three omnidirectional wheels that can reach the maximum speed of $1.8m/s$ (see Figure 1). The robot is equipped with a omnidirectional catadioptric vision sensor able to detect objects (e.g., ball, robots) up to a distance of about $6m$ all around the robot.

Preliminary experiments have been carried out on the task "go to ball", in which the robot must learn how to reach the ball as fast as possible. Although we have chosen a quite simple task, the large discretization adopted, the non-stationarity of the environment (due to battery consumption during the learning process), the noise affecting robot's sensors and actuators, and the limited amount of experience make the results obtained significant to evaluate the benefits of the PWC-Q_{LB}-learning algorithm. The state space of this task is characterized by two continuous state variables: the *distance* and the *angle* at which the robot perceives the ball. The ball distance has been discretized into five intervals: $[0:50), [50:100), [100:200), [200:350), [350:600]$, while the angle has been evenly split into 24 sectors 15° wide, thus obtaining a state space with 120 states. As far as the action space is concerned, thanks to its three omnidirectional wheels, the robot can move on the plane with three degrees of freedom mapped to three action variables:

- *module of the tangential velocity*, associated to the speed at which the robot translates. Its value is expressed in percentage of the maximum tangential velocity and it is discretized into 6 values $0, 20, 40, 60, 80, 100$;
- *direction of the tangential velocity*, the direction along with the robot translates. Its value is expressed in degrees and it is discretized into 24 evenly spaced values: $0^\circ, 15^\circ, 30^\circ, \ldots, 345^\circ$;
- *rotational velocity*, associated to the speed at which the robot changes its heading. Its value is expressed in percentage of the maximum rotational velocity and it is discretized into 9 values: $-20, -15, -10, -5, 0, 5, 10, 15, 20$.

The total number of available actions is $1,296$.

The reward function is such that the robot receives -1 as reward at each step except when its distance from the ball is below $50cm$ and the angle falls in the range $[-15^\circ : 15^\circ]$, in which case the reward is $+10$ and the trial ends. At the beginning of each learning trial, the robot starts form the center kickoff position and performs learning steps until it succeeds in reaching the ball which is positioned at $350cm$. Once a trial is finished, the learning process is suspended, and the robot autonomously performs a resetting procedure moving towards the starting position.

The sparsity of this reward function, although making easier its definition and preventing the introduction of biases, requires a long exploration period before catching some positive rewards and propagate the associated information to the rest of the state space. In this task, the robot would clueless wander around the field with little hope of success and high risk of bumping against objects around the field. For this reason, as mentioned in Section 2, we split the learning process into two phases. The first phase consists of "supervised" trials, i.e., trials in which the robot is controlled by hand-written behaviors and the RL algorithm only observes and records the actions taken in the visited states and the associated rewards. On the basis of the observed data, the RL algorithm builds a first approximation of the value function that will be exploited in the second phase. The acquired policy allows to make a safe exploration of the environment, thus considerably speeding up the learning process towards the optimal policy. In the second phase, the hand-written controller is bypassed by the RL system which chooses which action must be executed by the robot in each state, and, on the basis of the collected reward and the reached state, performs on-line updates of its knowledge.

In the following, we present comparative experiments among Q-learning, Q_{LB}-learning (see Section 3.3), and PWC-Q_{LB}-learning (see Section 3.4). The "supervised" phase has been performed only once and then the collected data have been reused in all the experiments. It consists of 60 trials with the ball placed in different positions around the robot within $400cm$. Since the actions produced by the

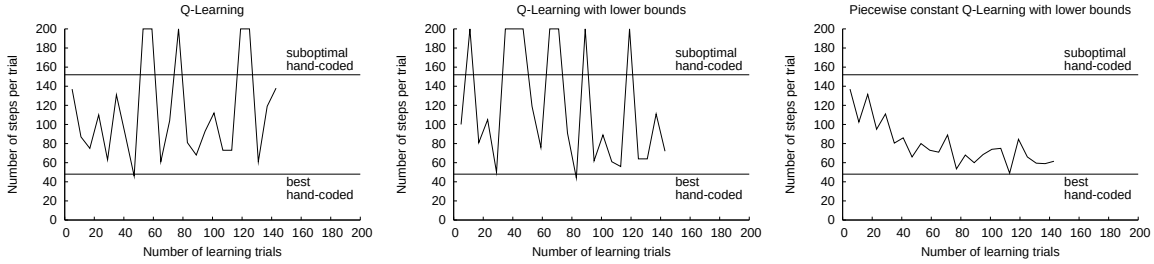

Figure 2: Performance of Q-learning, Q_{LB}-learning and PWC-Q_{LB}-learning on the "go to ball" task.

hand-written controller (based on fuzzy rules) are continuous, and given that the RL algorithms work with discrete actions, we replace each action produced by the controller with the one with the closest value among those in the discrete set of the RL system. Given the low complexity of the task, the hand-written behavior is, on purpose, highly suboptimal (with only low speed commands) in order to better highlight the improvements obtained by the learning processes in such a noisy environment.

In Figure 2 are reported the learning plots of the three algorithms during the second learning phase. The performance of the algorithms is measured by the number of steps (each step lasts 70*ms*) required to reach the ball. Every five trials the learning process is suspended for one trial; in this trial the robot executes the policy learned so far. Only the exploitation trials are shown in the graphs. When the robot is not able to reach the ball within 200 steps, the trial ends. For each graph are also reported the average performance of the suboptimal hand-coded policy followed by the controller ($\sim$ 152 steps) and the average performance of our best hand-coded policy ($\sim$ 48 steps). It is worth noting that, due to noise in the sensing and actuating systems and to the low accuracy of the resetting procedure, also the performance of fixed policies is affected by a high variance (standard deviation of about $\pm$10 steps). The parametrization is the same for each learning algorithm: the learning rate is 0.5 and decreases quite quickly, the ε-greedy exploration starts at 0.5 and decreases slowly, the discount factor is 0.99. The action values stored in the Q-table are initialized with a low value (-100), so that the values of the actions performed during the supervised trials become larger, thus biasing the exploration to regions near to the example trajectories. Given the stochasticity (due to sensor noise) and the non-stationarity (due to battery discharging), each experiment has been repeated three times for each algorithm, for a total of $1,350$ trials and more than $130,000$ steps.

In the first two graphs we have reported typical runs of Q-learning and Q_{LB}-learning. Both of them show a quick learning in the first trials, but then they alternate good trials to bad trials without appearing to converge to a stable solution. Trials that reach 200 steps typically mean that the robot, in some regions, has learned a policy that stops the movement of the robot. As explained above, this irrational behavior is due to self-updates (see Equation 8). Unsurprisingly, this problem is more frequent in the Q_{LB}-learning algorithm. The third plot displays the performance of PWC-Q_{LB}-learning algorithm averaged over three different runs. Here, the number of steps needed to reach the ball decreases quite slowly, but, unlike the other algorithms, the policy improves more and more reaching performance close to the best hand-coded policy, without any trial reaching the 200 steps limit.

It is worth briefly describing the policy learned by the robot with the PWC-Q_{LB}-learning algorithm: in the first trials the robot gradually learns to increase its speed in different situations until it learns to reach the ball at the maximum speed. Although this policy allows to complete some trials in very few steps (even less than 40 steps), it is likely to fail, since a coarse discretization gives control problems at high speed, especially in regions where a good accuracy is required. The problem is that, when the robot travels fast, it may not be able to go straight to the ball, and in general it hits the ball, thus needing to run after it for many steps. Given the risk aversion typical of minimax approaches, the PWC-Q_{LB}-learning algorithm quickly learns to give up with fast movements when the robot is near to the ball. The final result is that the robot starts at high speed and slows down when the ball gets closer, thus managing to reach the termination condition without touching the ball.

During one of the experiments another interesting characteristic of the PWC-Q_{LB}-learning algorithm emerged. After about 100 trials, one of the wheels began to partially lose its grip on the engine axis, thus causing the robot to turn slightly right instead of moving straightforward. Our learning algorithm managed

to face this non-stationarity by reducing the tangential speed in all the states, turning slightly right, and then it started to gradually raise the tangential speed. This behavior has put in evidence how this learning approach potentially can increase the robustness towards mechanical faults, thus increasing the autonomy of the robot. Further studies will be focused on a detailed analysis of this important aspect.

5 DISCUSSION AND FUTURE WORKS

The application of RL techniques to robotic applications could lead to the autonomous learning of optimal solutions for many different tasks. Unfortunately, most of the traditional RL algorithms fail when applied to real-world problems because of the time required to find the optimal solution, of the noise that affects both sensors and actuators, and of the difficulty to manage continuous state spaces.

In this paper, we have described and experimentally tested a novel algorithm (PWC-Q_{LB}-learning) designed to overcome the main issues that arise when learning is applied to real-world robotic tasks. PWC-Q_{LB}-learning computes the lower bound for the action value function while following a piecewise constant policy. Unlike other *min-max*-based algorithms, PWC-Q_{LB}-learning does not require the model of the dynamics of the environment and avoids long and blind exploration phases. Furthermore, it does not learn the optimal policy for the theoretically worst case, but it estimates the lower bound on the conditions actually experienced by the robot according to its current policy and to the current dynamics of the environment. Finally, the piecewise constant action selection and update guarantee a stable learning process in continuous state spaces, even when the discretization is such that the Markov property is lost.

Although preliminary, the experiments showed that PWC-Q_{LB}-learning succeeds in learning a nearly-optimal policy by optimizing the behavior of a sub-optimal controller in noisy continuous environments. Furthermore, it proved to be more stable with respect to Q-learning even when a coarse discretization of the state space is used. At the moment, we are currently investigating the theoretical properties of the proposed algorithm and we are testing its performance on more complex robotic tasks, such as the "align to goal" and the "kick" tasks.

REFERENCES

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton.

Bonarini, A., Matteucci, M., Restelli, M., and Sorrenti, D. G. (2006). Milan robocup team 2006. In *RoboCup-2006: Robot Soccer World Cup X*.

Buffet, O. and Aberdeen, D. (2006). Policy-gradient for robust planning. In *Proceedings of the Workshop on Planning, Learning and Monitoring with Uncertainty and Dynamic Worlds (ECAI 2006)*.

Gaskett, C. (2003). Reinforcement learning under circumstances beyond its control. In *Proceedings of International Conference on Computational Intelligence for Modelling Control and Automation*.

Heger, M. (1994). Consideration of risk in reinforcement learning. In *Proceedings of the 11th ICML*, pages 105–111.

Kitano, H., Asada, M., Osawa, E., Noda, I., Kuniyoshi, Y., and Matsubara, H. (1997). Robocup: The robot world cup initiative. In *Proceedings of the First International Conference on Autonomous Agent (Agent-97)*.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th ICML*, pages 157–163.

Millán, J. D. R. (1996). Rapid, safe, and incremental learning of navigation strategies. *IEEE Transactions on Systems, Man, and Cybernetics (B)*, 26(3):408–420.

Moore, A. and Atkeson, C. (1995). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21:711–718.

Morimoto, J. and Doya, K. (2000). Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. In *Proceedings of the 17th ICML*.

Morimoto, J. and Doya, K. (2001). Robust reinforcement learning. In *Advances in Neural Information Processing Systems 13*, pages 1061–1067.

Smart, W. D. and Kaelbling, L. P. (2002). Effective reinforcement learning for mobile robots. In *Proceedings of ICRA*, pages 3404–3410.

Sutton, R. S. (1996). Generalization in reinfrocement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38.

Watkins, C. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292.