Using Reinforcement Learning to Adapt an Imitation Task

Florent Guenter and Aude G. Billard

***Abstract*— The goal of developing algorithms for programming robots by demonstration is to create an easy way of programming robots that can be accomplished by everyone. When a demonstrator teaches a task to a robot, he/she shows some ways of fulfilling the task, but not all the possibilities. The robot must then be able to reproduce the task even when unexpected perturbations occur. In this case, it has to learn a new solution. In this paper, we describe a system that allows a robot to re-learn constrained reaching tasks by combining the knowledge acquired during the demonstration, with that acquired though reinforcement learning.**

I. INTRODUCTION

As robots start pervading human environments, the need for more natural human-robot interfaces is becoming more pressing. In particular, robots should be able to acquire new abilities through interactions with humans. Imitation learning is a promising mechanism for conveying new know-how, and is widely used in the animal kingdom [1]. Imitation learning has also been applied to robotics where it has also been called programming by demonstration [2].

In order to program a robot by demonstration, there are two main problems to solve. The first one is known as the "what to imitate" problem and consists in extracting the principal constraints that appear in a demonstrated task. The second is known as the "how to imitate" problem and consists in finding a solution to reproduce the demonstrated task despite of the difference of situation or embodiment .

In this paper, we address the "how to imitate" problem. More precisely, we investigate how a robot can adapt its execution of a learned task when confronted to a new situation. In our previous work [3], we have used dynamical systems (DS) to provide the required adaptation to changes in the task constraints, owing to the fact that DS are known to provide stable and robust control in uncontrolled environment [4]–[7]. However, DS work well only when the perturbations remain small. In the face of important changes in the task constraints, such as, for example, when an obstacle appears on the trajectory generated by the dynamical system, the DS fails at finding a solution. In those cases, an exploration process is needed during which the robot will search for new ways of accomplishing the task.

Reinforcement learning (RL) is particularly indicated for this type of problem, as the robot has to be able to improve its own capacity. Reinforcement learning algorithms have already been successfully implemented in robotic applications such as swinging up and controlling an inverse pendulum [8], refine forehand and backhand tennis swing [9] or acquiring a dynamic stand-up behavior [10], [11]. For all these examples, RL algorithm has been used to tune parameters in an equation that models the system's dynamic in a discrete environments. In order to use reinforcement learning on more complex robots like humanoid robots, the algorithms have to be adapted to tackle data lying in continuous time and space. Early works addressing this issue are those of Bradtke & Duff [12] on algorithms for continuous time semi-Markov Decision Problems and Doya's methods [13] such as Q-Learning or Actor-critic [14], [15] for continuous time and space dynamical systems. To deal with the high computing costs of the RL algorithms, function approximation techniques [16] and gradient descent techniques [17]–[19] have been proposed. Combining these techniques, Peters et al. [20], [21] proposed the natural actor-critic (NAC) algorithm based on natural gradient descent algorithms [22]. The NAC performs much better than classical gradient policy in term of convergence speed. In this paper, we extend our previous work to integrate the NAC to our algorithm for robot programming by demonstration using DS.

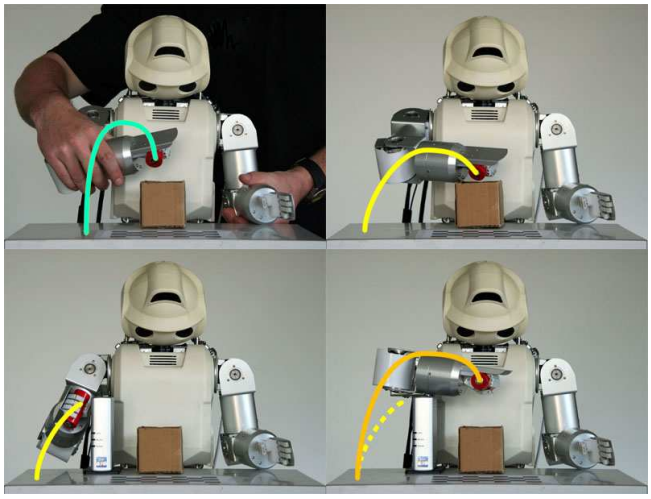

Fig. 1. Programming a robot by demonstration means that a user demonstrates a task and the robot has to extract the important features of the task in order to be able to reproduce it in different situations. It may happen in some special cases that the robot encounters a new situation where following closely the demonstration does not help to fulfill the task. In this case, there are two possibilities, the first one is to teach the robot how to handle the new situation and the second one is to let the robot learn by itself how to fulfill the task. In this paper we focus on the second possibility.

II. SYSTEM OVERVIEW

Let $\xi(\mathbf{t}) \in \mathbb{R}^s$ describe the complete state of the robot at each time step. In the application described in the rest of

F. Guenter and A.G. Billard are from the Learning Algorithms and Systems Laboratory, School of Engineering, Ecole Polytechnique Federale de Lausanne, Switzerland. florent.guenter@epfl.ch, aude.billard@epfl.ch

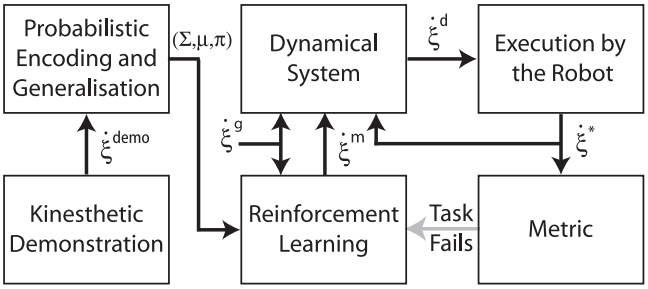

Fig. 2. Schematic overview of the complete system. In this paper, the variable ξ consists of the 4 joint angles θ of the robot's right arm. The speed trajectories $\dot{\xi}^{demo}$ provided by the set of demonstrations are fed into a learning system that trains a GMM model for the data. The value π, μ, Σ are used in a Reinforcement Learning module for generating a trajectory ξ^m that modulates a dynamical system with attractor ξ^g and output speed $\dot{\xi}^d$ executed by the robot. The effective trajectory measured on the robot is given by $\dot{\xi}^*$ and is analyzed using a metric. In our experiments, the metric requires that the distance between the target and the last point of the trajectory must be smaller than a certain value. If the task failed, we use a reinforcement learning algorithm which produces a new trajectory $\dot{\xi}^m$ in order to correct the modulation and fulfill the task.

this document, ξ consists of the 4 joint angles $\theta \in \mathbb{R}^n$ of the robot's right arm.

The aim of the algorithm is to reproduce the qualitative features common to several demonstrated trajectories, while being robust when the situation changes, for example, in the face of different initial conditions, target locations or obstacle on the path. The information flow of the algorithm is illustrated in Fig. 2. After having being exposed to several demonstrations $\dot{\xi}^{demo}(t)$ of the task, the algorithm extracts a generalized form of the original demonstration $\dot{\xi}^m(t)$ using a probabilistic model. The generalized trajectory is then used to modulate a DS which produces a trajectory to reach the target ξ^g.

When facing important perturbations, such as an obstacle blocking the arm of the robot, it may happen that the DS alone does not find a satisfying solution to reach the goal. To avoid that type of problem, we have implemented a reinforcement learning module which allows the robot to learn how to avoid the obstacles or other perturbations. The reinforcement learning acts directly on the modulation of the dynamical system. This way, the convergence properties of the DS are preserved. Note that the system described below does not make any assumption on the type of data and, thus, ξ could be composed of other variables, such as, for instance, the position of the robot's end effector or the data projected in a latent space as done in [23].

A. Probabilistic Encoding and Generalization

Learning of the task's constraints is done in a probabilistic manner, namely by encoding a set of demonstrated trajectories in a Gaussian Mixture Model (GMM) and by retrieving a *generalized* trajectory through Gaussian Mixture Regression (GMR [24]). Next, we briefly summarize this procedure, see [23], [25] for details. The method models the joint distribution of an "input" and an "output" variable as a Gaussian Mixture Model (GMM). In our case, the output variables are the velocities $\dot{\xi}$ and the input variable is the time t. If we join those variables in a vector $\upsilon = \begin{bmatrix} t \\ \dot{\xi} \end{bmatrix}$ it is possible to model its probability density function as

$$p(\upsilon) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\upsilon; \mu_k, \mathbf{\Sigma}_k) \tag{1}$$

where π_k is a weighting factor and $\mathcal{N}(\upsilon; \mu_k, \mathbf{\Sigma}_k)$ is a Gaussian function with mean μ_k and covariance matrix $\mathbf{\Sigma}_k$:

$$\mathcal{N}(\upsilon; \mu_k, \mathbf{\Sigma}_k) = \frac{1}{\sqrt{(2\pi)^d |\mathbf{\Sigma}_k|}} e^{(-\frac{1}{2}(\upsilon-\mu_k)^T \mathbf{\Sigma}_k^{-1}(\upsilon-\mu_k))} \tag{2}$$

where d is the dimensionality of the vector υ. The mean vector μ_k and covariance matrix $\mathbf{\Sigma}_k$ can be separated into their respective input and output components:

$$\mu_k = [\mu_{k,t}^T \ \mu_{k,\dot{\xi}}^T]^T \tag{3}$$

$$\mathbf{\Sigma}_k = \begin{pmatrix} \mathbf{\Sigma}_{k,t} & \mathbf{\Sigma}_{k,t\dot{\xi}} \\ \mathbf{\Sigma}_{k,\dot{\xi}t} & \mathbf{\Sigma}_{k,\dot{\xi}} \end{pmatrix} \tag{4}$$

This GMM can be trained using a standard E-M algorithm, taking the demonstrations as training data. We thus obtain a joint probability density function for the input and the output. Because it is a GMM, the conditional probability density function, i.e., the probability of the output conditioned on the input is also a GMM. Hence, it is possible, after training, to recover the expected output variable $\dot{\xi}^m$, given the observed input variable t. $\dot{\xi}^m$ is the modulation speed for the first trials $q = 1$

$$\dot{\xi}^m_{q=1}(t) = \sum_{k=1}^{K} h_k(t)\big(\mu_{k,\dot{\xi}} + \mathbf{\Sigma}_{k,\dot{\xi}t}\mathbf{\Sigma}_{k,t}^{-1}(t - \mu_{k,t})\big) \tag{5}$$

where the $h_k(t)$ are given by:

$$h_k(t) = \frac{\pi_k \mathcal{N}(t; \mu_{k,t}, \mathbf{\Sigma}_{k,t})}{\sum_{k=1}^{K} \pi_k \mathcal{N}(t; \mu_{k,t}, \mathbf{\Sigma}_{k,t})} \tag{6}$$

The variance of this conditional probability distribution is given by:

$$\mathbf{\Sigma}_{\dot{\xi}}(t) = \sum_{k=1}^{K} h_k^2(t)\big(\mathbf{\Sigma}_{k,\dot{\xi}} - \mathbf{\Sigma}_{k,\dot{\xi}t}\mathbf{\Sigma}_{k,t}^{-1}\mathbf{\Sigma}_{k,t\dot{\xi}}\big) \tag{7}$$

Thus, in our application, after training, the GMM can be used to generate a movement by taking the expected velocities $\dot{\xi}^m(t)$ conditioned on time t. This movement is taken to be the one to imitate. Moreover, the variances of the GMM can provide an indication about the variability of the observed variables. At any given time, variables with low variability across demonstrations can be interpreted as more relevant to the task than variables with high variability.

B. Dynamical System

The GMR signal is then coupled to a dynamical system to ensure that the task can still be reached under slight perturbations. The dynamical system that we use is inspired by the VITE model of human reaching movement, it can be associate to a spring and damper system. It allows the robot

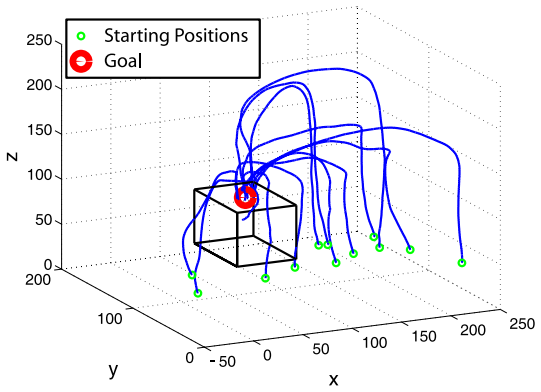

Fig. 3. In this example, the Gaussian Mixture Model is trained with 12 kinesthetic demonstrations of the task shown in Figure 1. Here, the demonstrations have been done with a fixed position for the box, but for the experiments presented in this paper, we have changed the position of the box for each demonstration.

to bring its arm smoothly to the target, following a straight trajectory (in the absence of perturbation).

$$\ddot{\xi}(t) = \alpha(-\dot{\xi}(t) + \beta(\xi^g - \xi^*(t))) \quad (8)$$

$$\dot{\xi}(t) = \dot{\xi}^*(t) + \ddot{\xi}(t) \quad (9)$$

The constants $\alpha, \beta \in \mathbb{R}_{[0,1]}$ are control parameters, $\dot{\xi}$ and $\ddot{\xi}$ are the current speed and acceleration of the DS, ξ^* is the current position of the robot and ξ^g represents the target position (the goal). Because there are some tasks for which the robot must follow a specific trajectory to reach the goal (for example putting a object in a box), we have to modulate the DS in order to fulfill these tasks. Our solution is to give to the robot a weighted average between the output of the dynamical system and a desired speed as command.

$$\dot{\xi}^d(t) = (1 - \gamma(t))\dot{\xi}(t) + \gamma(t)\dot{\xi}^m(t) \quad (10)$$

where $\dot{\xi}^m$ is the desired speed used to modulate the system and $\dot{\xi}^d$ is the speed command for the robot [1]. In order to ensure the convergence of the system on the target, we need to force the second term to zero at the end of the task. To realize this, we modulate $\dot{\xi}^m$ with the parameter γ which varies according to:

$$\gamma(t) = max(((T - t)/T)^2, 0) \quad (11)$$

where T is the duration of the observed movement and $\gamma(t) \in \mathbb{R}_{[0,1]}$ (see Fig. 4).

In the system described in [3], the modulation is given by Equ. 5. For the system described in this paper, this is available only for the first trial $q = 1$. If the target reaching fails, we have now the possibility to change the modulation using reinforcement learning.

[1]For security reason, we have implemented a system to avoid to reach the joint limits of the robot HOAP3. For that, we use a parameter L which is equal to 1 in most of each joint angle range and tends 0 when the joint angle approach the limits.

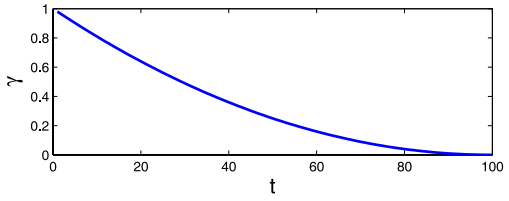

Fig. 4. Evolution of the parameter γ along time.

C. Reinforcement Learning

In order to allow the robot to find a new solution if the system fails to reach the goal, we have added in our system a reinforcement learning module. This module acts on the dynamical system through the modulation variable $\dot{\xi}^m(t)$ by optimizing the means $\mu_{k,\dot{\xi}}$ of Equ. 1.

The algorithm we have used for the reinforcement learning module of our system is based on the episodic natural actor-critic (NAC) algorithm presented in [20], [21]. The NAC is a variant of the actor-critic method for reinforcement learning. In the critic part of the NAC, the policy is evaluated by approximating the state-value function. For the approximation, an adaptation of a $LSTD(\lambda)$ (Least Square Temporal Difference) algorithm [16], [26] called $LSTD - Q(\lambda)$ [20] is used.

In the actor part, the policy is improved by using the "natural" gradient descent [22] which is a steepest gradient descent algorithm with respect to the Fisher information matrix. The first step is to define the policy of our system. In order to explore the space of the parameters, we introduce a stochastic policy defined by a Gaussian control policy:

$$\rho(\dot{\xi}^r, \dot{\xi}^m, \mathbf{\Sigma}_{\dot{\xi}}) = \frac{1}{\sqrt{(2\pi)^d |\mathbf{\Sigma}_{\dot{\xi}}|}} e^{(-\frac{1}{2}(\dot{\xi}^r - \dot{\xi}^m)^T \mathbf{\Sigma}_{\dot{\xi}}^{-1} (\dot{\xi}^r - \dot{\xi}^m))} \quad (12)$$

where $\dot{\xi}^m$ is the modulation speed of the dynamical system defined in Equ.5, $\mathbf{\Sigma}_{\dot{\xi}}$ is the covariance matrix (see Equ. 7) of the Gaussian control policy and $\dot{\xi}^r$ is the noisy command generated by $\rho(\dot{\xi}^r, \dot{\xi}^m, \mathbf{\Sigma}_{\dot{\xi}})$ and used to explore the parameters space. The parameters of the policy that we want to optimize are the means $\mu_{k,\dot{\xi}}$. By optimizing $\mu_{k,\dot{\xi}}$, we will be able to generate new trajectories $\dot{\xi}_q^m$ that will help the dynamical system to avoid the obstacle smoothly. The reward function used to evaluate the trajectory reproduced by the robot is defined as follows:

$$r_q(\dot{\xi}^s, \xi^s) = \sum_{t=0}^{T} -c_1 |\dot{\xi}_t^s - \dot{\xi}_{t,q=1}^m| - c_2 |\xi_T^s - \xi^g| \quad (13)$$

where $c_1, c_2 \in \mathbb{R}$ are weighting constants, ξ^s is the simulated state of the robot, $\xi_{t,q=1}^m$ is the modulation speed of the first trial (see Equ. 7) and ξ^g is the target position. Thus the reward function is determined by a term that represents the distance between the goal and the last point of the tested trajectory and a term that represents the similarity between

the current trajectory and the original modulation given by the GMM.

The following algorithm is applied separately on each $\mu_{k,\dot{\xi}}$. For ease of reading, we will thus omit the indice $_k$ in the following description. In the critic part of the algorithm, for the policy evaluation, like in most reinforcement learning algorithm, we evaluate the expected reward of a command $\dot{\xi}^r$ for a state $\dot{\xi}^s$. For that purpose as shown in [20], we can use the advantage function approximated by:

$$A^\rho(\dot{\xi}^r, \dot{\xi}^s) \approx \nabla_\pi ln\rho(\dot{\xi}^r, \dot{\xi}^m, \mathbf{\Sigma}_{\dot{\xi}})' \mathbf{w}_q \tag{14}$$

where $\mathbf{w}_q$ is a vector of weight that is equal to the approximation of the natural gradient of the expected return (see [20]). By using this approximation, we are then to evaluate of the reward on one trial:

$$\sum_{t=0}^{T} \gamma^t \nabla_\pi ln\rho(\dot{\xi}^r, \dot{\xi}^m, \mathbf{\Sigma}_{\dot{\xi}})' \mathbf{w}_q + \mathbf{v}_q \approx r_q \tag{15}$$

Where v_q represents the approximation of the value function at the initial state of each trial. To find the natural gradient $\mathbf{w}$, we will use a $LSTD - Q(\lambda)$ algorithm (define in [20]). For that, we have to define a new basis function:

$$\widehat{\phi} = \sum_{t=0}^{N} [1, \gamma^t \nabla_\Theta ln\rho(\dot{\xi}^r, \dot{\xi}^m, \mathbf{\Sigma}_{\dot{\xi}})']' \tag{16}$$

With this new basis function, we can rewrite Eq. 15 in a vectorial form:

$$\widehat{\phi} \cdot [\mathbf{v}_q, \mathbf{w}_q']' \approx r_q \tag{17}$$

For each trial q, we compute the basis function $\widehat{\phi}$ and the immediate reward r_q to update the sufficient statistics $\mathbf{A}_q$ and $\mathbf{b}_q$ (see [16]).

$$\mathbf{A}_q = \mathbf{A}_{q-1} + \widehat{\phi}_q \widehat{\phi}_q' \tag{18}$$

$$\mathbf{b}_q = \mathbf{b}_{q-1} + \widehat{\phi}_q r_q \tag{19}$$

These sufficient statistics are then used to update the critic parameters v_q and $\mathbf{w}_q$:

$$[\mathbf{v}_q, \mathbf{w}_q^T]' = \mathbf{A}_q^{-1} \mathbf{b}_q \tag{20}$$

After multiple trials, $\mathbf{w_q}$ converges to the natural gradient of the expected return. Thus, once w has converged over a window h, i.e. $\forall \tau \in [0, ..., h]$, the angle between w_{e+1} and $w_{e-\tau}) \leq \epsilon$, we are able to update the parameters $\mu_{\dot{\xi}}$ in order to optimize the expected return:

$$\mu_{q,\dot{\xi}} = \mu_{q-1,\dot{\xi}} + \alpha_{learn} \mathbf{w}_q \tag{21}$$

where $\alpha_{learn} \in \mathbb{R}_{[0,1]}$ is the learning rate. Once the parameters $\mu_{\dot{\xi}}$ are updated, we have to forget a part of the sufficient statistics $\mathbf{A}$ and $\mathbf{b}$ in order to approximate the new gradient $\mathbf{w}$.

$$\mathbf{A}_q \leftarrow \beta \mathbf{A}_q \tag{22}$$

$$\mathbf{b}_q \leftarrow \beta \mathbf{b}_q \tag{23}$$

where $\beta \in \mathbb{R}_{[0,1]}$. The algorithm converge then to an optimum with respect to the reward. In the experiments reported here, RL is stopped when the condition $|\xi_T^* - \xi^g| < \epsilon$ is satisfied.

III. EXPERIMENTS

The task chosen for testing the algorithm consists in putting an object into a box. The robot that we use is the humanoid robot HOAP3 from Fujitsu. It has a total of 25 DOFs, but for the manipulation tasks of this paper, we use only one 4 DOFs arm. The robot was first trained to do the task by demonstrating the task 26 times, starting in each case with a different initial arm posture and reaching for a different target location. During each demonstration, the robot recorded the trajectories followed passively by its 4 DOFs using the motor encoders. What the robot was expected to learn throughout the demonstration was that, no matter where the box was located, what mattered (i.e. what remained consistent throughout the demonstrations) was that the box should be reached from above (i.e. by looming over the box).

Once the robot has extracted the generalized speed profile $\dot{\xi}^m$ of the task, the robot is able to reproduce the task with various starting positions and for various locations of the box, by using only the dynamical system (DS) modulation (see [3]). In order to test the RL module of our algorithm, we have introduced an obstacle lying in the trajectory of the robot's end effector. In this case, the DS modulation fails to bring the robot's hand to the target. The RL module helps then the robot reach the goal by avoiding the obstacle while trying to still satisfy the constraints of the movement shown during the demonstrations. The tests of the RL module are done in simulation.

A. Results

This section provides the results of experiments aimed at evaluating the abilities of the RL module.

1) The whole system: Fig. 5 represents a typical example of what we want to obtain with the reinforcement learning. The target (represented by a small circle) lies on the top of the cubic box and the obstacle is represented by the thinner box. The obstacle is placed in the middle of the trajectory generated using the modulated dynamical system without any learning (dashed line). By using the reinforcement learning module, the system has been able to find a solution (solid line) that avoids the obstacle. We observe in Fig. 5 that it took around 750 trials in simulation to find a solution. We can see that the reward function tends to an optimum, but we do not observe the convergence because the system interrupts the learning process as soon as it obtains a satisfying solution for ξ^*. In figure 5, we can observe the effect of the interruption on ξ^m (the thin line), ξ^m avoid the obstacle but do not reach the goal.

2) The stochastic exploration process: In order to study the convergence property of the RL algorithm, we have conducted a series of tests to have statistics on the number of trials we need to converge to a solution. The box and the obstacle have the same position for all the experiments, only the starting position is moving. We have defined 23 starting positions equally distributed along a vertical line. For each of these positions, we have made 12 runs of the RL algorithm combined with the dynamical system. To limit

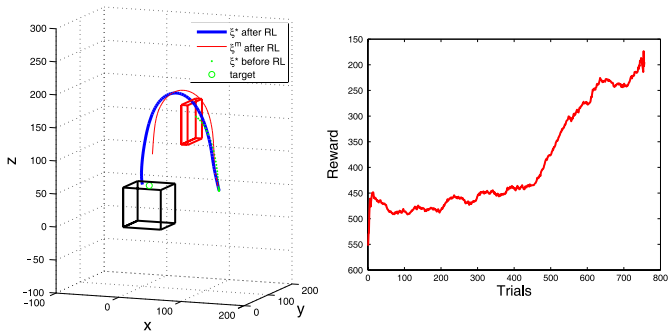

Fig. 5. **On the left:** The dynamical system alone would have produced the trajectory represented by the dashed line. By using the reinforcement learning module, the algorithm is able to find a different trajectory that avoids the obstacle (the thick line). The learning have been interrupted before the convergence because a satisfying solution has been found. **On the right:** This Graphic represents the evolution of the reward function. We observe that, for this example, it takes around 750 steps for the algorithm to find a solution. The learning is interrupted as soon as the dynamical system finds a satisfying solution.

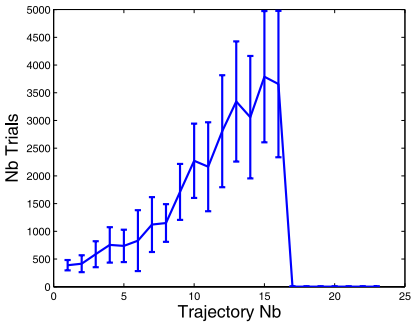

Fig. 6. This graphic represents the statistics of the number of trials needed to converge to a solution. As abscissa, we have the starting position (23 starting positions equally distributed along a vertical line, the position number 1 is the lowest and the number 23 is the uppermost position). As ordinate, we have the mean and standard deviation of the number of trials for each run of the RL algorithm. We observe here that the solution seems to be more difficult to obtain with the lower starting points.

the computation time, we have limited the maximum number of steps to $10'000$ steps. Fig. 6 represents the statistics (mean and standard deviation) for the 23 starting positions. Position number 1 is the lower position on the vertical line while position 23 is the upper (see sample trajectories in Fig. 7). We can see on Fig. 6 that the RL algorithm has more difficulties to find a solution for the middle starting positions. This is certainly due to the fact that it is more difficult to find a way while keeping the bell shape of the demonstrated trajectory. For position 17 and bigger, the DS is sufficient to find a solution.

This phenomenon can also be observed in Fig. 7. This Fig. represents the different trajectories for 6 chosen starting positions. A interesting point is that for position 16 (see Fig.7), the algorithm find a better way by passing aside the obstacle and not above or under.

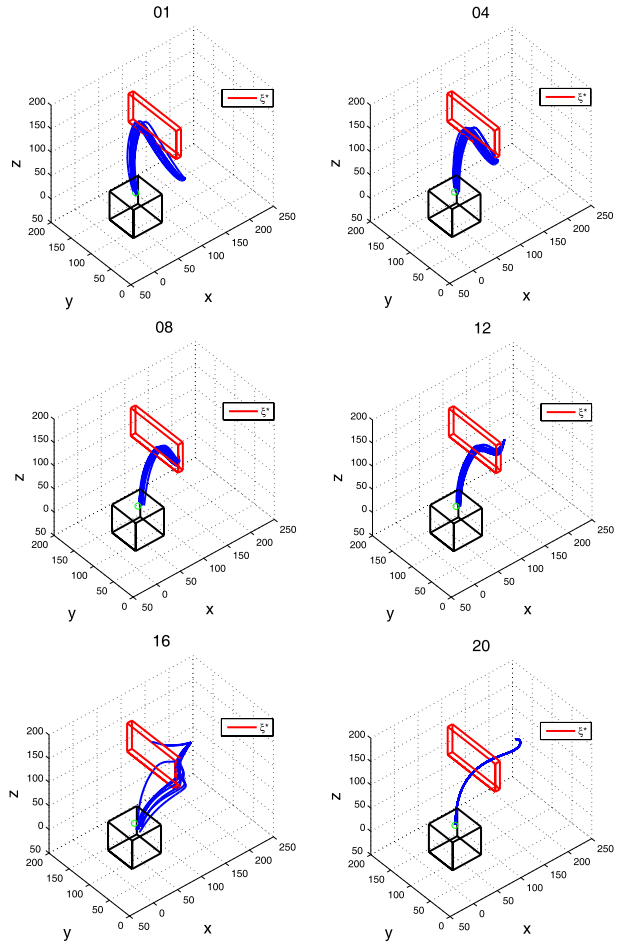

Fig. 7. We observe here the different trajectories for 6 selected starting positions. The number of each graph correspond to a position represented in Fig. 6. The number 01 represents the lower starting point and the 23 the upper. We have not represented the trajectories for which the RL module haven't found a solution within the $10'000$ trials.

IV. DISCUSSIONS AND CONCLUSIONS

We have presented in this paper an algorithm for learning and reproducing goal-directed tasks with a humanoid robot. At the first attempt to reproduce the task, the trajectory is generated using a dynamical system modulated by a velocity profile generated by a GMM trained with a few demonstrations performed by the user. If there are some perturbations that can not be handled by the dynamical system alone, a reinforcement learning method can be used in order to adapt the means of the GMM that encode the modulation speed, and therefore find an other smooth solution.

A limitation of the current system is that it requires to relearn completely the trajectory for each new obstacle, because the learning is done on a modulation that does not depend on the obstacle. In future work, we will implement the reinforcement learning on parameters that are in relation with the obstacle, in order to have a better solution, valid even if the obstacle moves.

Currently the reproduction part of the system is done in simulation. We are currently working on adapting this

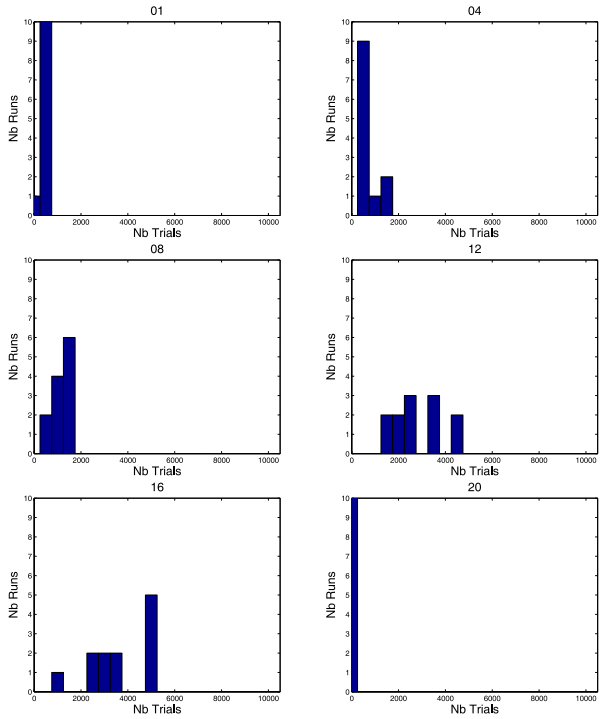

Fig. 8. These figures represents the histogram of the number of trials needed to converge to a solution for the 6 starting position shown in Fig. 7. During the experiment, the number of trials where limited to $10'000$, we observe here that there are for each starting points at least one runs where the RL algorithm has not found a satisfying solution. Like in Fig. 7, we see that this problem is more important in the lower starting positions.

algorithm on the real robot. However, the system generally needs more than 1000 trials to converge to a solution that avoids the obstacle. It seems unrealistic to do the learning on the real robot since this model require at least 6 seconds for a single trial. Because we are working with quasi-static movements, we expect that it will be possible to keep the learning part of the process in simulation in order to shorten the time to find a solution. To have a more realistic simulation, we will use Webots[2], a commercial mobile robot simulation software developed by Cyberbotics Ltd.

While there is still room for improvements, the results presented here strengthen the idea that imitation should not be considered as a passive repetition of a task, but rather as an active process, in which exploration plays a significant and often underestimated role. This suggest that reinforcement learning is a suitable paradigm for implementing such an explorative process.

REFERENCES

[1] K. Dautenhahn and C. Nehaniv, Eds., *Imitation in Animals and Artifacts*. The MIT Press, 2001.

[2] A. Billard and R. Siegwart, Eds., *Robotics and Autonomous Systems, Special Issue: Robot Learning From Demonstration*. Elsevier, 2004, vol. 47, no. 2,3.

[3] M. Hersch, F. Guenter, S. Calinon, and A. Billard, "Learning dynamical system modulation for constraint reaching tasks," in *IEEE-RAS International Conference on Humanoid Robots-HUMANOIDS'06*, 2006.

[4] G. Schoner, M. Dose, and C. Engels, "Dynamics of behaviour: Theory and application for autonomous robot architecture," *Robotics and Autonomous Systems, vol. 16, pp. 213-245*, 1995.

[5] M. Hersch and A. Billard, "A biologically-inspired model of reaching movements," in *IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics*, 2006.

[6] I. Iossifidis and G. Schoner, "Autonomous reaching and obstacle avoidance with anthropomorphic arm of a robotics assistant using the attractor dynamics approach," in *IEEE International Conference on Robotics and Automation*, 2004.

[7] L. Righetti and A. Ijspeert, "Programmable central pattern generators: a application to biped locomotion control," in *IEEE International Conference on Robotics and Automation*, 2006.

[8] C. Atkeson and S. Schaal, "Learning tasks from a single demonstration," in *Proceedings of the IEEE/RSJ Int. Conference on Robotics and Automation (ICRA'97)*, vol. 2, 1997.

[9] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, "Learning movement primitives," in *International Symposium on Robotics Research (ISRR2003), Springer Tracts in Advanced Robotics*, 2004.

[10] J. Morimoto and K. Doya, "Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning," *Robotics and Autonomous Systems, vol. 36, pp. 37-51*, 2001.

[11] S. Iida, M. Kanoh, S. Kato, and H. Itoh, "Reinforcement learning for motion control of humanoid robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[12] S. Bratke and M. Duff, "Reinforcement learning methods for continuous-time markov decision problems," in *Neural Information Processing Systems Conference*, 1994.

[13] K. Doya, "Reinforcement learning in continuous time and space," *Neural Computation, vol. 12, pp. 219-245*, 2000.

[14] R. S. Sutton and A. G. Barto, *Reinforcement learning : an introduction*, ser. Adaptive computation and machine learning. Cambridge, Mass. : MIT Press, 1998.

[15] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in Neural Information Processing Systems 12. MIT Press*, 2000.

[16] A. Nedic and D. Bertsekas, "Least-squares policy evaluation algorithms with linear function approximation," *LIDS Report LIDS-P-2537, Dec. 2001; to appear in J. of Discrete Event Systems*, 2002.

[17] R. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning, 8, pp. 229-256*, 1992.

[18] R. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in Neural Information Processing Systems 12. MIT Press*, 2000.

[19] A. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems: Theory and Applications, 13, pp. 343-379*, 2003.

[20] J. Peters, S. Vijayakumar, and S. Schaal, "Reinforcement learning for humanoid robotics," in *Humanoids2003, Third IEEE-RAS International Conference on Humanoid Robots*, 2003.

[21] ——, "Natural actor-critic," in *16th European Conference on Machine Learning, pp. 280-291*, 2005.

[22] S. Amari, "Natural gradient works efficiently in learning," *Neural Computation, vol. 10, pp. 251-276*, 2000.

[23] S. Calinon, F. Guenter, and A. Billard, "On learning, representing and generalizing a task in a humanoid robot," *IEEE Transactions on Systems, Man and Cybernetics, 37:2. Part B. Special issue on robot learning by observation, demonstration and imitation*, 2007.

[24] Z. Ghahramani and M. Jordan, "Supervised learning from incomplete data via an em approach," *Advances in Neural Information Processing Systems 6*, 1994.

[25] A. Billard, S. Calinon, and F. Guenter, "Discriminative and adaptative imitation in uni-manual and bi-manual tasks," *Robotics and Autonomous Systems, vol. 54*, 2006.

[26] J. Boyan, "Technical update: Least-squares temporal difference learning," *Machine Learning, 49, pp. 233-246*, 2002.

[2]http://www.cyberbotics.com