# Reinforcement Learning for Imitating Constrained Reaching Movements

Florent Guenter, Micha Hersch, Sylvain Calinon and Aude Billard

*LASA Laboratory - Ecole Polytechnique Federale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland*

{*florent.guenter, micha.hersch, sylvain.calinon, aude.billard*}*@epfl.ch*

**Abstract**

The goal of developing algorithms for programming robots by demonstration is to create an easy way of programming robots such that it can be accomplished by anyone. 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 to teach to the robot constrained reaching tasks. Our system is based on a dynamical system generator modulated by a learned speed trajectory. This system is combined with a reinforcement learning module to allow the robot to adapt the trajectory when facing a new situation, for example in the presence of obstacles.

*keywords*: Programming by Demonstration, Reinforcement Learning, Dynamical Systems, Gaussian Mixture Model.

## 1 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 [4].

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 inspite of the difference of embodiment or situation [1]. 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 with a new situation. The robustness of dynamical system control to perturbations occurring in uncontrolled environments has

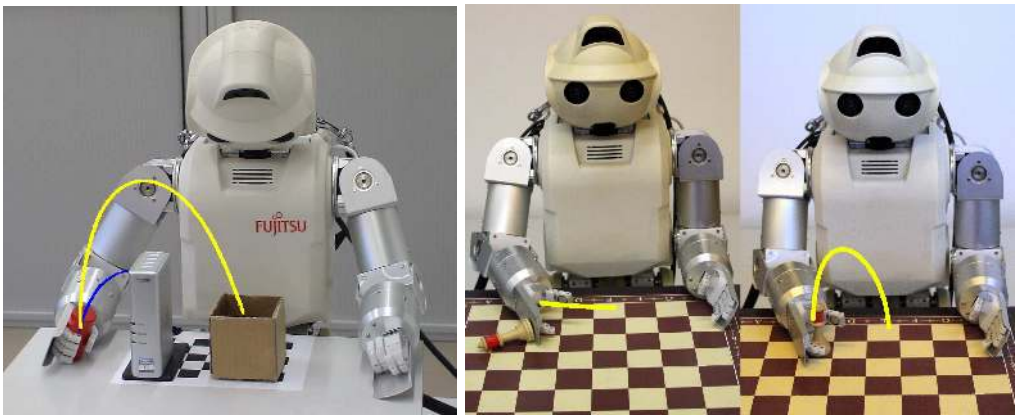


Figure 1: Programming a robot by demonstration means that a user demonstrates a task and the robot has to extract the important features of the task in order to be able to reproduce it in different situations. It might happen in some special cases that the robot encounters a new situation where the demonstration 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 implement the second possibility.

been demonstrated [5, 18–20]. However, it can happen that the dynamical system alone is not sufficient to handle the perturbation (for example when an obstacle appears on the trajectory generated by the dynamical system). 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 it enables the robot to learn its own new ways of acting, in order to improve its abilities. Reinforcement learning algorithms have already been successfully implemented in robotic applications such as swinging up and controlling an inverse pendulum [21], refining forehand and backhand tennis swing [22] or acquiring a dynamic stand-up behavior [6, 23]. Reinforcement learning algorithms perform well in discrete environments of small dimensionality. However, to use reinforcement learning on real robots, the algorithms have to be adapted to tackle data of high dimensionality lying in continuous time and space. Bradtke & Duff [24] have derived algorithms for continuous time semi-Markov Decision Problems, Doya [7] proposed methods such as Q-Learning or Actor-Critic [2, 8] for continuous time and space. To deal with the high computing costs of the RL algorithms for higher dimensionality systems, Peters et al. [25, 26] proposed the Natural Actor-Critic (NAC) algorithm which is based on function approximation and gradient descent algorithms [9–13] More recently, studies on the effect of the reward function on the speed of learning [27], on the effect of using inaccurate model [28] or on new efficient exploration process [29] have been conducted in the RL field. In this paper, we will integrate a RL module in the dynamical system that we have developed in [30] in order to allow the robot to learn new solutions if the demonstrations do not provide a sufficient solution for the reproduction of the skill. This RL module will be based on Jan Peters' work on the NAC [26].

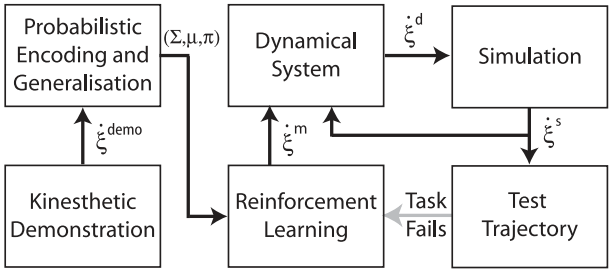


Figure 2: Schematic overview of the complete system. In this paper, the variable $\xi$ contains the joint angle of the robot arm. The speed trajectories $\dot{\xi}^{demo}$ provided by the set of demonstrations are fed into a learning system that trains a Gaussian Mixture Model (GMM) in order to build a probabilistic model of the data. The parameters of the GMM are used in a Reinforcement Learning module for generating a trajectory $\xi^m$ that modulates a dynamical system with attractor $\xi^g$ and output speed $\dot{\xi}^d$ executed by the robot. The simulation of the trajectory reproduced by the robot is given by $\dot{\xi}^s$ and is tested to know if it reaches the target or not. This is done by measuring the distance between the target and the last point of the trajectory, this distance must be smaller than a given value to validate the task. If the task fails, we use a reinforcement learning algorithm which produces a new trajectory $\dot{\xi}^m$ in order to correct the modulation and fulfill the task.

## 2 System Overview

Let $\xi(t) \in \mathbb{R}^s$ describe the complete state of the robot at each time step. In the application described in the rest of this document, $\xi$ consists of the joint angles of the robot 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 obstacles 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 demonstrations $\dot{\xi}^m(t)$ using a probabilistic model. The generalized trajectory is then used to modulate a dynamical system which produces a trajectory to reach the target $\xi^g$. When facing important perturbations, such as an obstacle blocking the arm of the robot, it may happen that the dynamical system alone does not find a satisfying solution to reach the target. 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 dynamical system are preserved. Note that the system described below does not make any assumption on the type of data and, thus, $\xi$ 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 [14].

## 2.1 Probabilistic Encoding and Generalization

In this section, we briefly summarize the Gaussian Mixture Regression (GMR [15]) procedure used to obtain a single "model" trajectory from several demonstrations. This application has been described in details in [14, 16]. The principle of this method is to model the joint distribution of an "input" and "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 $\pi_k$ is a weighting factor and $\mathcal{N}(\upsilon; \mu_k, \mathbf{\Sigma}_k)$ is a Gaussian function with mean $\mu_k$ 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 $\upsilon$. The mean vector $\mu_k$ 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 $t$.

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

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. This method is illustrated in Fig. 3 on the left graphic, where one sees a set of trajectories (light gray lines),

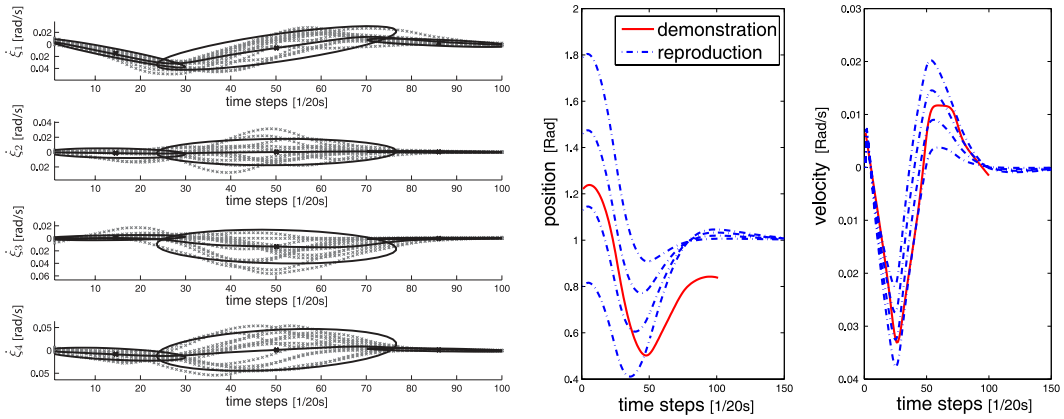


Figure 3: **Left:** This graphic represents the Gaussian Mixture Model (described in 2.1) trained with the 26 kinesthetic demonstrations of the the box task. $\xi_1$ to $\xi_4$ are the four joint angles of the robot arm. The thin lines represent the demonstrations and the thick lines the generalization $\dot{\xi}^m$ retrieved by using the GMR. The ellipses represent the Gaussian components of the joint probability distribution. **Right:** This graphic represents different trajectories generated using the modulated dynamical system (described in 2.2). The reproduced trajectories (dash-dotted line) are qualitatively similar to the modulation trajectory $\xi^m$ (solid line), although they reach the goal from different initial positions.

the Gaussian mixture components modeling them (ellipses) and the expected trajectory (thick line). 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.

## 2.2 Dynamical System

The dynamical system that we use is a spring and damper system inspired by the VITE model of human reaching movement (see [18]). It allows the robot to bring its arm smoothly to the target, following a straight trajectory (in the absence of perturbation).

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

$$\dot{\xi}(t) = \dot{\xi}^s(t) + \ddot{\xi}(t)\Delta t \tag{9}$$

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 dynamical system, $\xi^s$ is the simulated current position of the robot and $\xi^g$ 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 output of the dynamical system 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 modulation speed as command.

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

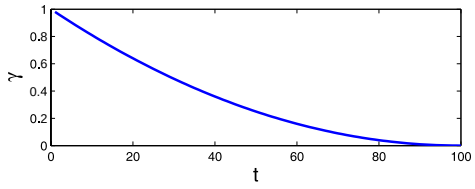


Figure 4: Evolution of the parameter $\gamma$ along time.

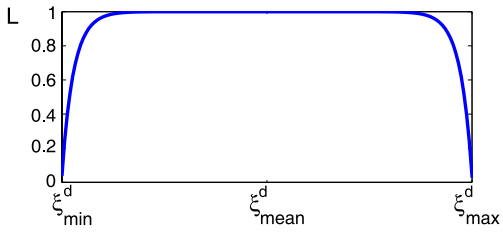


Figure 5: Evolution of the parameter $L$ in function of $\xi^d$.

where $\dot{\xi}^m$ is the desired speed used to modulate the system and $\dot{\xi}^d$ is the speed command for the robot. 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 $\gamma$ which varies according to:

$$\gamma(t) = ((T-t)/T)^2 \tag{11}$$

Where $T$ is the duration of the observed movement and $\gamma(t) \in \mathbb{R}_{[0,1]}$. Eq. 11 ensure a smooth decay to zero for the modulation term $\dot{\xi}^m(t)$ (see Fig. 4)

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 $L$ in order to decrease the speed amplitude when we approach of the joint limits. The position update is done as follow:

$$\xi^d(t+1) = \xi^d(t) + L\dot{\xi}^d(t)\Delta t \tag{12}$$

where

$$L = \begin{cases} F(\xi^d_t) & \text{if } \xi^d(t) \in [\xi^d_{min}, \xi^d_{max}] \\ 0 & \text{if } \xi^d(t) \notin [\xi^d_{min}, \xi^d_{max}] \end{cases} \tag{13}$$

and

$$F(\xi^d) = 1 - \mid arctanh(1.52 * (\frac{\xi^d(t) - \xi_{mean}}{\xi^d_{max} - \xi^d_{min}}))^{10} \mid \tag{14}$$

The shape of the $L$ parameter within the joint limits $[\xi^d_{min}, \xi^d_{max}]$ is shown in Fig. 5.

In the system described in [30], the modulation $\xi^m$ is given by Eq. (5). For the system described in this paper, this is available only for the first trial $q = 1$ ($q$ denotes the trial number). During this first trial, if the system fails to reach the target, the modulation can be changed by using the RL module.

## 2.3 Reinforcement Learning

In order to allow the robot to find a new solution, when it fails to reach the target, 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 Eq. (1).

By default, for the first trial $q = 1$ (where $q$ denotes the trial number), the means $\mu_{k,\dot{\xi}}$ are given by the GMM model. Thus, we avoid interferences of the reinforcement learning module if the dynamical system is sufficient to ensure the reproduction of the task. If the system fails to reach the target by using the initial GMM, the RL is applied to learn a new set of means $\mu_{k,\dot{\xi}}$ for the GMM. The whole learning process is done in simulation even though we will implement the system on the robot. It is indeed not realistic to make 100 or 200 trials with the robot in a real environment. It is much more efficient to simulate the process and to find a solution after a few seconds. The advantage of Reinforcement Learning techniques over other direct path planning techniques is that it allows the robot to handle different type of situation. The robot is able to learn by exploring the solution space and discovering new solutions.

The algorithm we have used for the reinforcement learning module of our system is the episodic natural actor-critic (NAC) algorithm presented in [25,26]. 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 [9, 17] called $LSTD - Q(\lambda)$ [25] is used (see Appendix). In the actor part, the policy is improved by using the "natural" gradient descent [13] which is a steepest gradient descent algorithm with respect to the Fisher information matrix. The use of approximation functions and "natural" gradient descent makes this algorithm efficient enough to deal with multiple DOFs systems.

The first step is to define the policy of our system. In order to explore the space of the parameters, we introduce a stochastic Gaussian control policy:

$$\rho(\dot{\xi}^r, \dot{\xi}^m, \mathbf{\Sigma}_{\dot{\xi}}) = \frac{1}{\sqrt{(2\pi)^d |\mathbf{\Sigma}_{\dot{\xi}}|}} e^{(-\frac{1}{2}(\dot{\xi}^r - \dot{\xi}^m)^T \mathbf{\Sigma}_{\dot{\xi}}^{-1}(\dot{\xi}^r - \dot{\xi}^m)))}, \tag{15}$$

where $\dot{\xi}^m$ is the modulation speed of the dynamical system defined in Eq. (5), $\mathbf{\Sigma}_{\dot{\xi}}$ is the covariance matrix (see Eq. (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. We consider here that the demonstrations performed by the user are sufficiently informative to allow the robot to reproduce the task in standard conditions. We thus choose to keep the covariance matrix $\mathbf{\Sigma}_{\dot{\xi}}$ in order to respect the constraints taught by the demonstrator during the exploration process of the RL module.

The parameters of the policy that we want to optimize are the means $\mu_{k,\dot{\xi}}$ that represent the means of the GMM model. By learning new means $\mu_{k,\dot{\xi}}$, we will be able to generate new trajectories $\dot{\xi}_q^m$ that will help the dynamical system to avoid the obstacle smoothly.

The episodic reward function used to evaluate the produced trajectory is defined as follows:

$$r_q(\dot{\xi}^r, \xi^r) = \sum_{t=0}^{T} -c_1|\dot{\xi}_t^r - \dot{\xi}_{t,q=1}^m| - c_2|\xi_T^r - \xi^g|, \tag{16}$$

where $c_1 > 0, c_2 > 0 \in \mathbb{R}$ are weighting constants, $\xi^r$ is the simulated noisy command used to explore the solution space, $\xi_{t,q=1}^m$ is the modulation speed of the first trial (see Eq. (7)) and $\xi^g$ is the target position. Thus the reward function is determined by a term that represents the similarity between the current trajectory and the original modulation given by the GMM and a term that represents the distance between the target and the last point of the tested trajectory.

The effect of the first term on the reward function is that $r_q$ tends to a maximum $r_q < 0$ in most of the cases. Because the trajectory $\xi_{t,q=1}^m$ is the expected output of the GMM and does not depend on the target, it does not reach the target in most of the cases. Thus, after the learning, if the target is reached, there is still a difference between the current trajectory $\xi_{t,q}^m$ and the initial trajectory $\xi_{t,q=1}^m$ due to the deformation needed to reach the target.

The effect of the second term on the reward function is that $r_q$ tends to a maximum when $\xi_{t,q}^m$ reaches the target. The obstacle is implicitly represented in this term. During the learning, when the arm is blocked by a obstacle, the last point of the trajectory $\xi_T^s$ does not reach the target, thus the second term of $r_q$ does not reach zero with a obstacle on the trajectory.

The following algorithm is applied separately on each $\mu_{k,\dot{\xi}}$ (described in 2.1). For ease of reading, we will thus omit the index $_k$ in the following description. In the critic part of the algorithm, for the policy evaluation, like in most RL algorithms, we evaluate the expected reward of a command $\dot{\xi}^r$ for a state $\dot{\xi}^s$. That is generally done by the evaluation of the Action-Value function defined as:

$$Q^\rho(\dot{\xi}^r, \dot{\xi}^s) = E\{\sum_{t=0}^{\infty} \gamma^t r_t | \dot{\xi}^r, \dot{\xi}^s\} \tag{17}$$

where $\gamma \in \mathbb{R}_{[0,1]}$ is a discount factor and $r_t$ is the immediate reward. Note that this model is for a non episodic task. In the same way, we can define the Value function as:

$$V^\rho(\dot{\xi}^s) = E\{\sum_{t=0}^{\infty} \gamma^t r_t | \dot{\xi}^s\} \tag{18}$$

As shown in [25], these two expressions allow us to define an advantage function:

$$A^\rho(\dot{\xi}^r, \dot{\xi}^s) = Q^\rho(\dot{\xi}^r, \dot{\xi}^s) - V^\rho(\dot{\xi}^s), \tag{19}$$

where $A^\rho(\dot{\xi}^r, \dot{\xi}^s)$ represents the advantage of action $\dot{\xi}^r$ over the state $\dot{\xi}^s$. To adapt the NAC to an episodic task, we formulate the discounted sum of the advantage function along one trial as:

$$\sum_{t=0}^{T} \gamma^t A^\rho(\dot{\xi}^r, \dot{\xi}^s) = r_q(\dot{\xi}^r, \xi^r) + \sum_{t=0}^{T} \gamma^{t+1} V^\rho(\dot{\xi}_{t+1}^s) - V^\rho(\dot{\xi}_t^s) \tag{20}$$

$$= r_q(\dot{\xi}^r, \xi^r) + \gamma^{T+1} V^\rho(\dot{\xi}_{T+1}^s) - V^\rho(\dot{\xi}_0^s) \tag{21}$$

Note that for a episodic task, $\gamma^{T+1} V^\rho(\dot{\xi}_{t+1}^s) = 0$. In order to have an efficient algorithm we use a linear approximation function for $V^\rho(\dot{\xi}^s)$:

$$V^\rho(\dot{\xi}^s) \approx \phi(\dot{\xi}^s)' \mathbf{v}_q, \tag{22}$$

where $\mathbf{v}_q \in \mathbb{R}^T$ is a vector of weights and $\phi(\dot{\xi}^s)' \in \mathbb{R}^T$ is the transpose of a feature vector. In our case, since only $V^\rho(\dot{\xi}_0^s)$ in Eq. (21) has to be evaluated, we need only one basis function to approximate $V^\rho(\dot{\xi}^s)$. Thus, we can arbitrary set $\phi(\dot{\xi}^s) = 1$ and approximate the value function at time $t = 0$ by using only the weight $v_q$.

Following [25], we approximate the advantage function with:

$$A^\rho(\dot{\xi}^r, \dot{\xi}^s) \approx \nabla_\mu ln\rho(\dot{\xi}^r, \dot{\xi}^m, \mathbf{\Sigma}_{\dot{\xi}})^T \mathbf{w}_q, \tag{23}$$

where $\mathbf{w}_q$ is a vector of weights that is equal to the approximation of the natural gradient of the expected return. By introducing the approximation presented in Eq. (21), Eq. (22) and Eq. (23), we obtain:

$$\sum_{t=0}^{T} \gamma^t \nabla_\mu ln\rho(\dot{\xi}^r, \dot{\xi}^m, \mathbf{\Sigma}_{\dot{\xi}})^T \mathbf{w}_q + v_q \approx r_q(\dot{\xi}^r, \xi^r) \tag{24}$$

To find the natural gradient $\mathbf{w}$ we will use a $LSTD - Q(\lambda)$ algorithm (see Appendix). For that, we have to define a new basis function by using Eq. (18) and Eq. (23):

$$\widehat{\phi} = \sum_{t=0}^{N} \gamma^t [1, \nabla_\mu ln\rho(\dot{\xi}^r, \dot{\xi}^m, \mathbf{\Sigma}_{\dot{\xi}})^T]^T \tag{25}$$

With this new basis function, we can rewrite Eq. (24) in a vectorial form:

$$\widehat{\phi} \cdot [v_q, \mathbf{w}_q^T]^T \approx r_q(\dot{\xi}^r, \xi^r) \tag{26}$$

By using $LSTD - Q(\lambda)$ (see Appendix) we are then able to approximate $\mathbf{w}$ and $v$. For each trial $q$, we compute the basis function $\widehat{\phi}$ and the episodic reward $r_q(\dot{\xi}^s)$ to update the sufficient statistics $\mathbf{A}_q$ and $\mathbf{b}_q$ (see [9]).

$$\mathbf{A}_q = \mathbf{A}_{q-1} + \widehat{\phi}_q \widehat{\phi}_q^T \tag{27}$$

$$\mathbf{b}_q = \mathbf{b}_{q-1} + \widehat{\phi}_q r_q \tag{28}$$

At the initialization, $\mathbf{A}_1$ is set to $\mathbf{I}$ and $\mathbf{b}_1$ is set to $\mathbf{0}$. These sufficient statistics are then used to update the critic parameters $v_q$ and $\mathbf{w}_q$:

$$[v_q, \mathbf{w}_q^T]^T = \mathbf{A}_q^{-1} \mathbf{b}_q \tag{29}$$

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 $w_{q+1}$ and $w_{q-\tau} \leq \epsilon$, we are able to update the parameters[1] $\mu_{\mathbf{q},\dot{\xi}}$ in order to optimize the expected return:

$$\mu_{q,\dot{\xi}} = \mu_{q-1,\dot{\xi}} + \alpha_{learn} \mathbf{w}_q, \tag{30}$$

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 $\mathbf{w}$.

$$\mathbf{A}_q \leftarrow \beta \mathbf{A}_q \tag{31}$$

$$\mathbf{b}_q \leftarrow \beta \mathbf{b}_q, \tag{32}$$

[1] For each Gaussian of the GMM separately.

where $\beta \in \mathbb{R}_{[0,1]}$. The algorithm converges then to an optimum with respect to the reward.

The goal of the RL part of the system is to produce a trajectory $\dot{\xi}^m$ that will be used to modulate the dynamical system according to Eq. (10). Thus, at each update of the means $\mu_{k,\dot{\xi}}$, we test the new trajectory $\xi^s$ produced by the dynamical system, in order to stop the learning as soon as a satisfying solution is found. Because we are, for the time being, only interested in goal-directed reaching movements, we test the distance between the last point of the trajectory $\xi_T^s$ and the target. The task is considered as fulfilled if $|\xi_T^s - \xi^g| < d$ where $d \in \mathbb{R}$ represents the maximal distance we want to obtain.

# 3 Experiments

We have chosen two tasks for testing the algorithm. The first one consists in putting an object into a box and the second one consists in grasping a chess queen on a table. 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 the 4 DOFs arm. To generate $\dot{\xi}^m$ by training the GMM, we have performed twenty-six kinesthetic demonstrations on the robot for each task. It means that, for each demonstration, the demonstrator moved the robot arm to perform the task. The robot recorded the trajectories of its 4 DOFs using its own encoders. The specification of the demonstration for the first task was to raise the hand in order to reach the box from above. For the second task, the specification was to grasp the chess queen following either a vertical movement (to grasp it from above) or a horizontal movement that is constraint by the need of a specific direction.

Once we have extracted the generalized speed $\dot{\xi}^m$ of the first task, by using only the dynamical system described in 2.2, the robot is able to reproduce the task with various starting points and for various locations of the box (see [30]). However, the dynamical system alone is not able to find satisfying solutions for the avoidance of possible obstacles obstructing the trajectory. In those experiments, we have introduced such an obstacle. When the obstacle lies in the trajectory of the end effector of the robot, the reinforcement learning module helps the robot reach the goal by avoiding the obstacle while trying to still satisfy the constraints of the movement shown during the demonstrations (i.e. reaching the goal with a bell-shaped trajectory).

For the second task, the constraint to fulfill is focused on the approach direction, in order to grasp the chess queen without knocking it down. To model this constraint in simulation, we use a forbidden volume around the chess queen. It means that, if the center of the hand penetrates this volume, we consider that the chess queen is knocked down and thus that the reproduction of the task has failed. The reinforcement learning is then used to grasp the chess queen without entering the forbidden volume.

## 3.1 The Box Task

This section provides the results of experiments aimed at evaluating the abilities of the RL module. In the first part (3.1.1), the whole system is tested and the learning is stopped when a satisfying solution

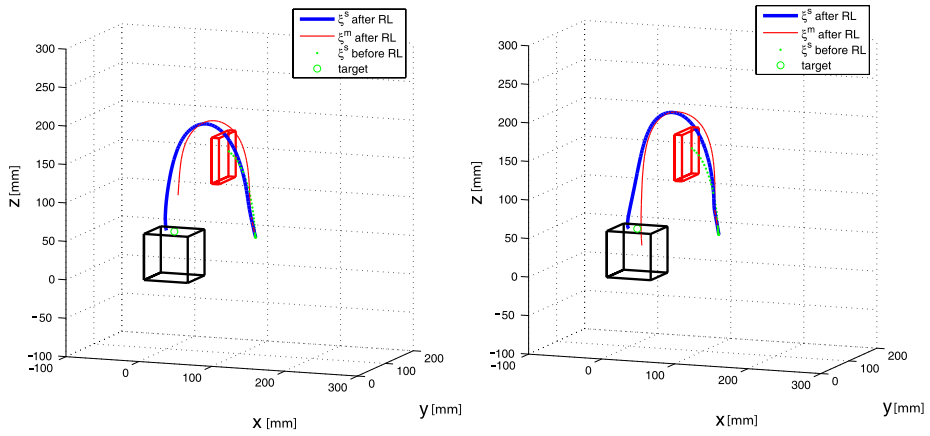


Figure 6: **Left:** The dynamical system alone would have produced the trajectory represented by the dotted line. By using the reinforcement learning module, the algorithm is able to find a different trajectory that avoids the obstacle (in thick line). The trajectory produced by the modulation speed $\dot{\xi}^m$ (in thin line) does not reach the goal. The learning have been interrupted before the convergence because a satisfying solution has been found for $\xi^d$. **Right:** Here, we can see that the trajectory produced by $\dot{\xi}^m$ reaches the goal. In this example, we use a convergence criterion to stop the learning, in order to show the convergence of the reinforcement learning module.

has been found for $\xi^s$ (as stated in 2.3). In the second part (3.1.2), we will use an other criterion to determine the end of the learning, a convergence criterion. In this case, the learning is stopped only when an optimal solution has been found for $\xi^m$. This is equivalent to a system which would reproduce a trajectory only by reinforcement learning without using a dynamical system (in other words, we do not consider $\xi^d$).

### 3.1.1 The whole system

Fig. 6 on the left 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 for $\xi^d$ (solid line) that avoids the obstacle. We observe in Fig. 7 that it takes around 330 trials in simulation to find a solution. We can see that the reward function tends to an optimum, but the learning is interrupted before the complete convergence of the system (as soon as a satisfying solution for $\xi^d$ is found). In Fig. 6 on the left, we can observe the effect of the interruption on the modulation $\xi^m$ (the thin line), $\xi^m$ avoids the obstacle but does not reach the goal.

To verify that the system is not sensitive to the particular choice of the obstacle location, we have conducted a set of simulations whereby the obstacle position varies along a vertical plan that crosses

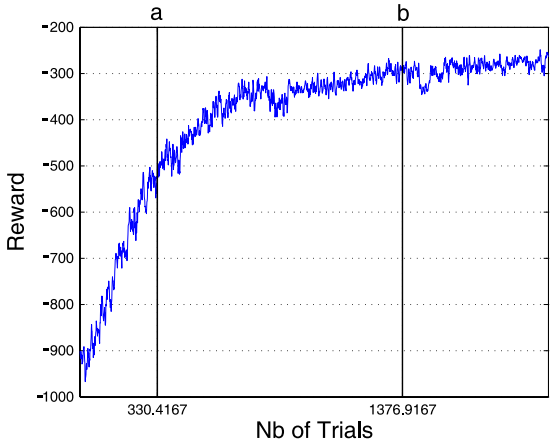


Figure 7: This Graphic represents the evolution of the reward function during the learning phase of the two examples shown in Fig. 6. This curve is a mean over 12 runs of the system in identical situations. The first vertical line (a) represents the average number of trials needed when the learning is interrupted as soon as a satisfying solution is found (see 3.1.1). The second vertical line (b) represents the average number of trials needed when the learning is interrupted by a convergence criterion as described in 3.1.2.

the trajectory. We thus obtain a mapping of the number of trials needed by the RL module to find a solution as a function of the obstacle location. Because of the stochastic process used to explore the parameters space, we have used the average on a few runs of the RL module at each point to have a good representation of the different areas where the system has difficulties to find some solutions. Fig. 8 represents this mapping for an average on four runs at each point. We can observe that it can find a solution for most locations in less than 300 steps. In simulation, it represents approximatively 40s when the system runs on a PC equipped with a Pentium 4, 2.4 GHz processor. The area which still causes some problem to the system is when the obstacle is placed at the lowest position. This can be explained by the fact that the system tends to find a solution by passing under the obstacle (this is due the reward function). Along the $Y$ axis of the robot, we also observe more problems when the obstacle gets far from the robot. This is due to the fact that the setup is not centered in the middle of the robot workspace, thus when $Y$ increases the robot tries to pass aside the obstacle and gets stuck because of its joint angles limits.

### 3.1.2 The RL module

To show the convergence of the RL system (represented by $\xi^m$), we have changed the criterion to stop the learning. We use here a convergence criterion, when $r_q(\xi^s)$ has converged over a window $h_r$ (i.e. $\forall \tau \in [0, ..., h]$, the mean difference between $r_q$ and $r_{q-\tau} \leq \epsilon_r$), the learning is stopped. By using the convergence criterion, the learning continues until $\xi^m$ reaches the goal. That can be observed in Fig. 6 on the right. In Fig. 7, we observe the convergence of the reward function to an optimum around $-300$ which is a higher value than in the first example, but we also need a average of 1370 trials to converge

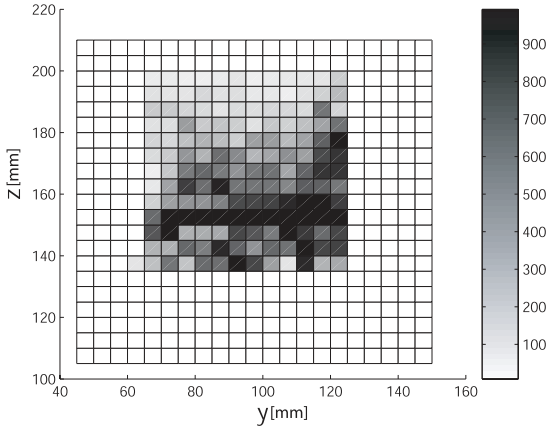


Figure 8: This figure represents the average number of trials needed by the reinforcement learning module to find a solution to reach the target in function of the middle point coordinate of the obstacle.

to this optimum.

By comparing the two graphics in Fig. 6, we can see that using the distance to the target or a convergence criterion does not influence significantly the final result for $\xi^s$, but rather influences the time needed to find a satisfying solution (see Fig. 7).

The other advantage to use the distance to the target as stopping condition can be seen in the case of a trajectory that is not suddenly blocked by an obstacle. In this case, the distance criterion do not enable the learning. If the criterion for stopping the learning is a convergence criterion, the learning is done in every case even when a solution already exists.

### 3.1.3 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 that 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 22 runs of the RL algorithm without the help of the dynamical system. To limit the computation time, we have limited the maximum number of steps to $10'000$ steps.

Fig. 9 represents the statistics (mean and standard deviation) for the 23 starting positions. Position number 1 is the lowest position on the vertical line while position 23 is the uppermost (see sample trajectories in Fig. 10). We can see on Fig. 9 that the RL algorithm has more difficulties to find a solution for the lower starting position. This is certainly due to the fact that it is more difficult to keep the bell shape of the demonstrated trajectory while finding a way between the obstacle and the box than satisfying the same bell shape by passing above the obstacle.

This phenomenon can also be observed in Fig. 10. These graphics represent the different trajectories

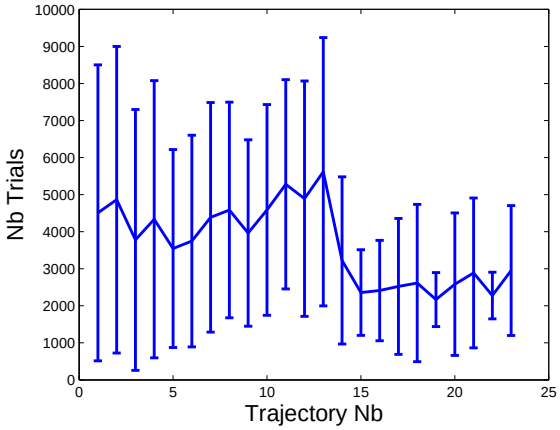


Figure 9: This graphic represents the statistics of the number of trials needed to converge to a solution. As abscissa, we have the starting position (23 starting positions equally distributed along a vertical line, the position number 1 is the lowest and the number 23 is the uppermost position). As 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.

for 6 chosen starting positions [2]. The big standard deviations we observe in Fig. 9 for the lower positions are due to the failure of the RL algorithm.

### 3.2 The Chess Task

For this experiment, we have modified the episodic reward function in order to include a negative reward if the reproduced trajectory penetrates the forbidden volume $V$.

$$r_q(\dot{\xi}^s, \xi^s) = \sum_{t=0}^{T} -c_1|\dot{\xi}_t^s - \dot{\xi}_{t,q=1}^m| + c_2 P(\dot{\xi}_{t,q}^m) - c_3|\xi_T^s - \xi^g| \tag{33}$$

where

$$P(\dot{\xi}_{t,q}^m) = \begin{cases} -1 & \text{if } \dot{\xi}_{t,q}^m \in V \\ 0 & \text{if } \dot{\xi}_{t,q}^m \notin V \end{cases} \tag{34}$$

We have conducted the same series of experiments as for the Box Task. The position of the queen is fixed and we have chosen 23 starting positions distributed along a horizontal line in front of the robot. The first position is situated in front of the robot and the last position in the right. The task is reproduced using the right arm. For this experiment, the stopping condition is that $\xi_q^s$ has avoided $V$ and that $\xi_{T,q}^s$ is on the target. Fig. 12 represents the trajectories for 6 selected starting positions. We observe that when the starting position is not in front of the queen, the preferred solution is to grasp the queen from above, but we observe some exceptions for starting positions 1, 9, 17, and 21.

[2]The trajectories for which the RL algorithm has not found a satisfying solution within the 10′000 trials are not shown in those graphics.

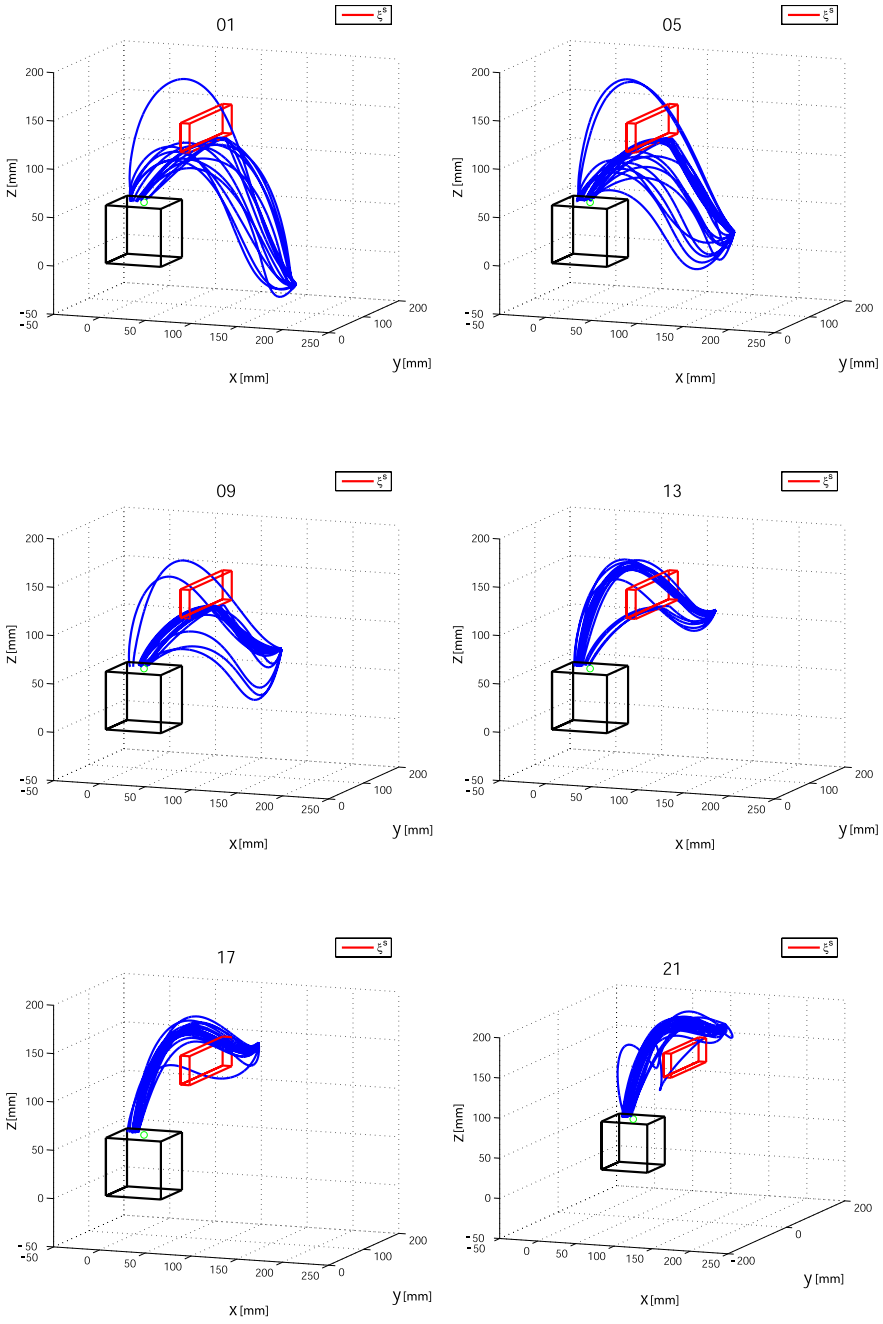


Figure 10: We observe here the different trajectories for 6 selected starting positions. The number for each graph corresponds to a position represented in Fig. 9. The number 01 represents the lowest starting point and the 23 the uppermost. We have not represented the trajectories for which the RL module haven't found a solution within the 10′000 trials.

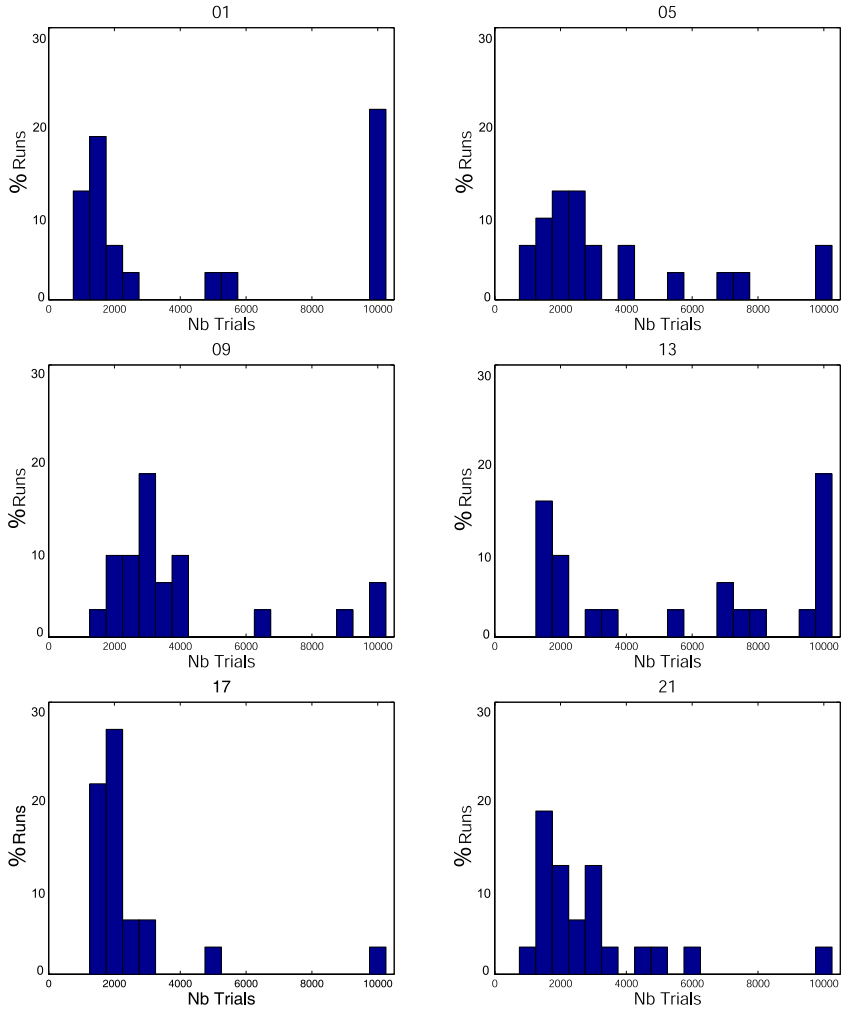


Figure 11: These figures represents the histogram of the number of trials needed to converge to a solution for the 6 starting position shown in Fig. 10. As the number of trials was limited to 10′000. We observe here that, for each starting point, there is at least one run where the RL algorithm has not found a satisfying solution. Like in Fig. 10, we see that this problem is more important in the lower starting positions.

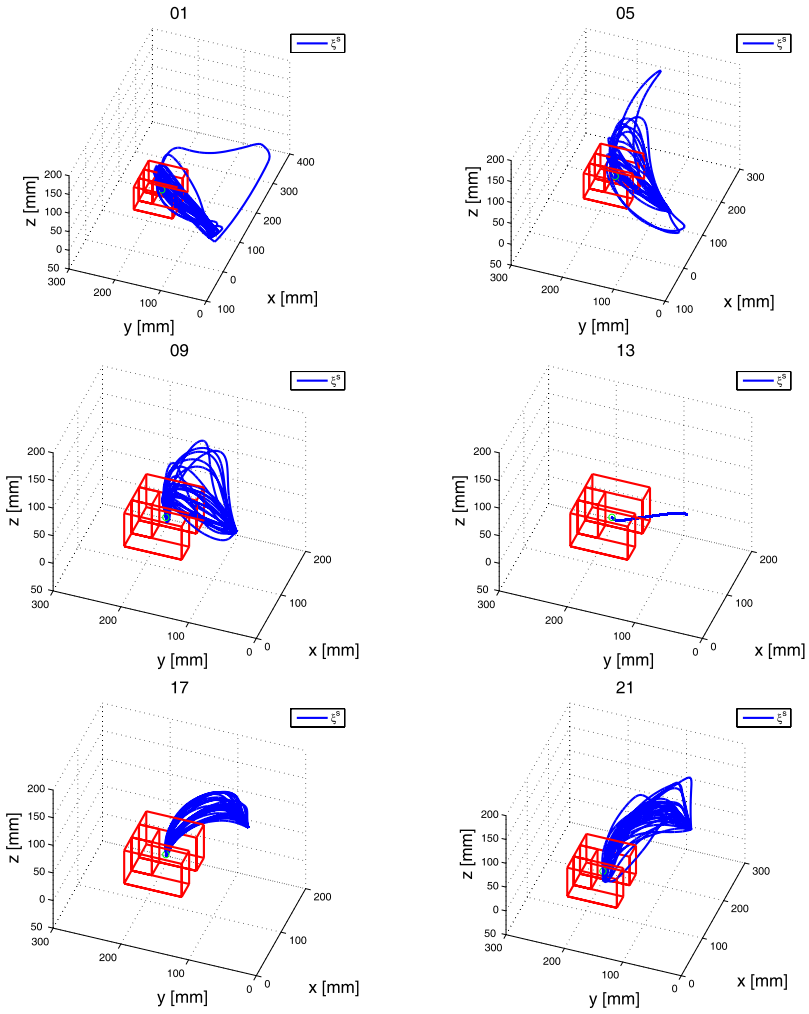


Figure 12: This figure represents the trajectories for 6 starting point of the chess task.

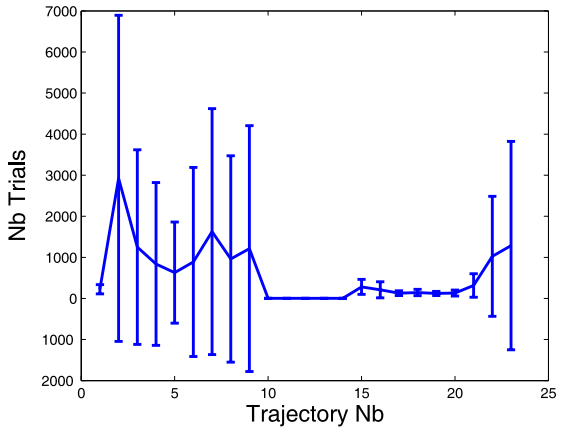


Figure 13: This graphic represents the statistics of the number of trials needed to converge to a solution for the chess task. As abscissa, we have the starting position (23 starting positions equally distributed along a horizontal line from the center to the right of the robot). As ordinate, we have the statistics on the number of trials needed to find a solution depending on the starting position.

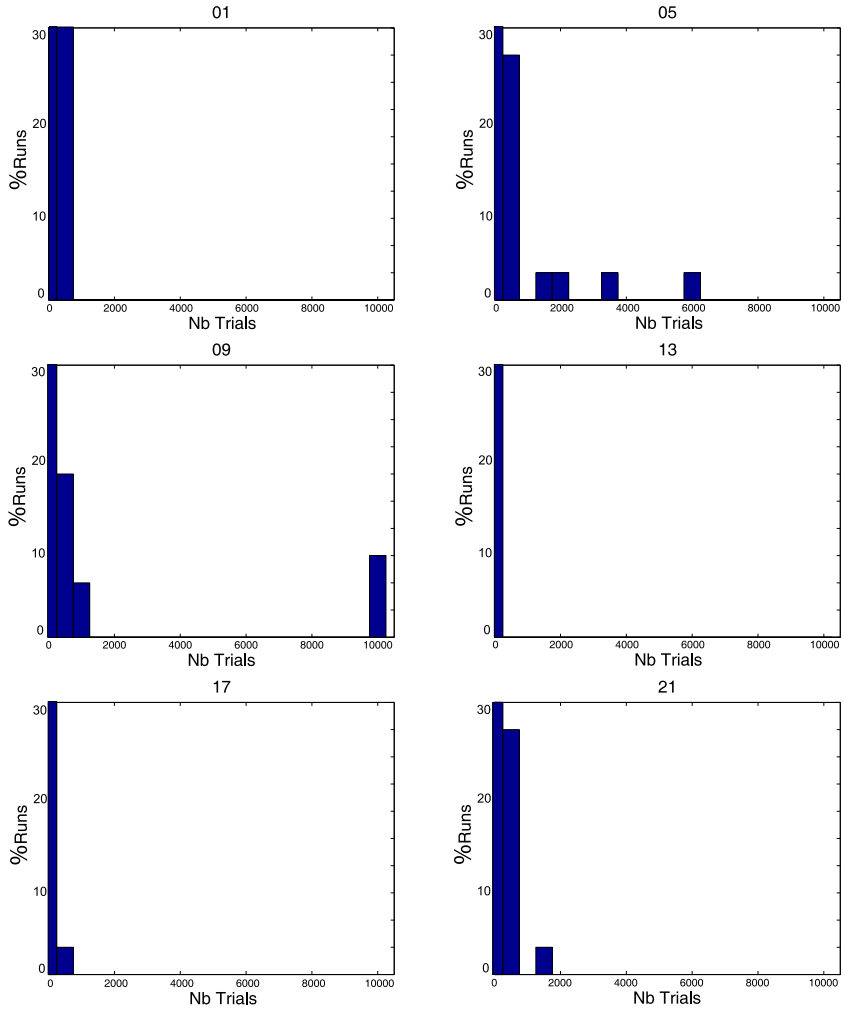


Figure 14: This figure represents the histogram of the number of trials for each run for 6 selected starting positions of the chess task.

Fig. 13 represents the statistics of 11 runs of the system for each starting point. As in the box task, we observe cases of failure of the algorithm. This phenomenon is more accentuated for the leftmost positions (from 1 to 9). This is due to the fact that we are near the joint angle limit for these starting positions. For starting positions 10 to 14, the task is reproduced correctly at the first trial. These starting positions are in front of the chess queen and then the dynamical system is sufficient for grasping the queen. For starting position on the right, reinforcement learning is necessary but a solution is found in a small number of trials, except for the two last positions that are near the joint limits.

# 4 Discussion

In Section 3, we have compared two ways of using reinforcement learning. The first one consisted in evaluating the trajectory $\xi^s$ and in interrupting the learning as soon as $\xi^s$ reaches the goal. The second solution was to learn the new $\xi^m$, to evaluate the RL until $\xi^m$ reaches the goal while avoiding the

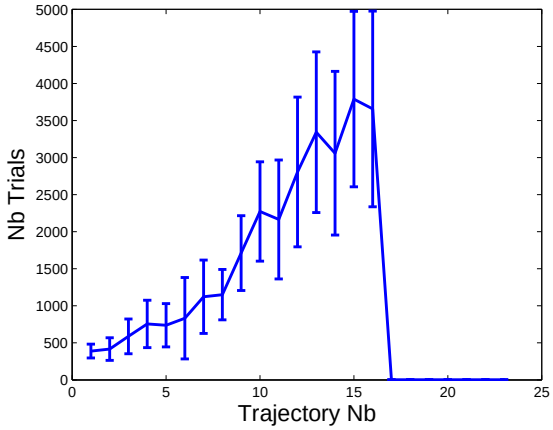


Figure 15: This graphic represents the statistics of the number of trials needed to find a solution with the dynamical system associated to the RL algorithm. We observe that the mean increases when the starting point goes up to point 16. For point 17 and upper, the obstacle is no more on the trajectory and the system finds a solution at the first attempt.

obstacle.

From the reinforcement learning point of view, the second solution is the best solution because we converge to an optimum for the modulation speed $\dot{\xi}^m$. But for our purpose, it is not needed. As shown in section 3.1.2, the resulting trajectory $\xi^s$ is similar, but the convergence to an optimum takes more time, when learning the optimal $\xi^m$. We have seen in Section 3.1.1 that it takes around $40s$ to do 300 trials. By comparing Fig. 9 and Fig. 15, we observe important differences in the mean value of the trials number needed to find a solution with the two systems. We have a total mean of 3495 trials for the RL alone and a total mean of 1253 for the combination of the dynamical system and the RL algorithm. The mean time needed by the two systems to find a solution while running on Matlab on a PC equipped with a Pentium 4, 2.4 GHz processor is $7min$ $46s$ for the RL alone and $2min$ $47s$ for the combination of the two.

This great difference in the mean computation time is mainly due to the cases where there is no obstacle on the original trajectory. The dynamical system is sufficient to find a trajectory that reaches the goal, thus the distance criterion validates the solution at the first attempt. If we choose to impose the optimum convergence to stop the learning, we will have many learning steps (around 2500 as shown in Fig. 9 for point 16 to 23), which will slow down the process even when the situation does not require it. In other words, for each reproduction, even the simplest ones, we will have to wait a mean of $5min$ $33s$ before moving.

A limitation of the system is concerned with the function $\gamma$ that managed the importance of the modulation along time. If we want to guarantee that the system converges to the goal, $\gamma$ has to be equal to zero at the end of the movement. For this paper, $\gamma$ is a fixed function and the influence of the modulation decreases rapidly with $t$. Thus, if we have an obstacle at the end of the trajectory, the

current system may not find a solution. In further work, we will investigate ways to learn the function $\gamma$ and the possibility to use the RL without the dynamical system so solve such situations.

# 5 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 is used in order to adapt the centers of the GMM components that encode the modulation speed, and therefore find an other smooth solution.

We have tested the system on two different tasks. In the first task, an obstacle perturbs the trajectory and the system has to find a new solution. In the second task, the goal is to grasp a chess piece. Due to the morphology of the robot, it cannot grasp the chess piece from every direction. Here the system can be used to optimize the task. The advantage of the reinforcement learning is that it can handle a lot of different and unexpected situations.

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. They show that reinforcement learning is a suitable paradigm for implementing such an explorative process.

# Acknowledgment

The work described in this paper was partially conducted within the EU Integrated Project COGNIRON ("The Cognitive Companion") and funded by the European Commission Division FP6-IST Future and Emerging Technologies under Contract FP6-002020, and EU Project IST-2004-004370 RobotCub (ROBotic Open-architecture Technology for Cognition, Understanding, and Behaviour). It was also supported by the Swiss National Science Foundation, through grant 620-066127 of the SNF Professorships program.

# REFERENCES

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

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning : an introduction*. MIT Press, Cambridge, MA (1998).

[3] D. Bertsekas and J. Tsitsiklis, *Neuro-dynamic programming*. Athena Scientific, Belmont, MA (1996).

[4] A. Billard and R. Siegwart, *Robotics and Autonomous Systems, Special Issue: Robot Learning From Demonstration*, Elsevier, Amsterdam(2004).

[5] 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).

[6] 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).

[7] K. Doya, Reinforcement Learning in Continuous Time and Space. *Neural Computation*, vol. **12**, pp. 219-245 (2000).

[8] V.R. Konda and J.N. Tsitsiklis, Actor-Critic Algorithms. *Adv. in Neural Information Process. Syst.*, **12**, pp. 1008-1015 (2000).

[9] A. Nedic and D. Bertsekas, Least-Squares Policy Evaluation Algorithms with Linear Function Approximation. *LIDS Report LIDS-P-2537, Dec. 2001*, (2001).

[10] R. Williams, Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, vol. **8**, pp. 229-256 (1992).

[11] R.S. Sutton, D. McAllester, S. Singh, and Y. Mansour, Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Advances in Neural Information Processing Systems*, **12**, (2000).

[12] A.G. Barto and S. Mahadevan, Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems: Theory and Applications*, vol. **13**, pp. 343-379 (2003).

[13] S. Amari, Natural Gradient Works Efficiently in Learning. *Neural Computation*, vol. **10**, pp. 251-276 (2000).

[14] S. Calinon, F. Guenter and A. Billard, On Learning, Representing and Generalizing a Task in a Humanoid Robot. *IEEE Trans. on Sys. Man and Cybernet. Part B. (Special issue on robot learning by observation, demonstration and imitation)* **37**(2) (2007).

[15] Z. Ghahramani and M.I. Jordan, Supervised Learning from Incomplete Data via an EM Approach. *Advances in Neural Information Processing Systems, Morgan Kaufmann Publishers*, vol. **6** (1994).

[16] A. Billard, S. Calinon and F. Guenter, Discriminative and adaptative imitation in uni-manual and bi-manual tasks. *Robotics and Autonomous Systems*, **54:5** (2006).

[17] J.A Boyan, Technical Update: Least-Squares Temporal Difference Learning. *Machine Learning*, vol. **49**, pp. 233-246 (2002).

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

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

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

[21] C.G. Atkeson and S. Schaal,Learning tasks from a single demonstration, in *Proc. IEEE International Conference on Robotics and Automation*, Albuquerque, (1997).

[22] S. Schaal, J. Peters, J. Nakanishi and A. Ijspeert, Learning Movement Primitives, in *Proc. International Symposium on Robotics Research (ISRR2003)*, Siena, (2003).

[23] S. Iida, M. Kanoh, S. Kato and H. Itoh, Reinforcement Learning for Motion Control of Humanoid Robots, in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, (2004).

[24] S.J. Bratke and M.O. Duff, Reinforcement Learning Methods for Continuous-Time Markov Decision Problems, in *Proc. Neural Information Processing Systems Conference*, Denver, (1994).

[25] J. Peters, S. Vijayakumar and S. Schaal, Reinforcement Learning for Humanoid Robotics, in *Proc. IEEE-RAS International Conference on Humanoid Robots (Humanoids2003)*, Muenchen, (2003).

[26] J. Peters, S. Vijayakumar and S. Schaal, Natural Actor-Critic, in *Proc. 16th European Conference on Machine Learning*, Porto, (2005).

[27] G. Konidaris and A. Barto, Autonomous Shaping: Knowledge Transfer in Reinforcement Learning, in *Proc. International Conference on Machine Learning*, Pittsburgh, (2006).

[28] P. Abbeel and M. Quigley, Using Inaccurate Models in Reinforcement Learning, in *Proc. International Conference on Machine Learning*, Pittsburgh, (2006).

[29] O. Simsek and A. Barto, An intrinsic Reward Mechanism for Efficient Exploration, in *Proc. International Conference on Machine Learning*, Pittsburgh, (2006).

[30] M. Hersch, F. Guenter, S. Calinon and A. Billard, Learning Dynamical System Modulation for Constraint Reaching Tasks, in *Proc. IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS06)*, Genova, (2006).

# APPENDIX

In this appendix we summarize the $LSTD(\lambda)$ algorithm presented in [9, 17] and the $LSTD-Q(\lambda)$ derived from $LSTD(\lambda)$ in [25].

To understand $LSTD(\lambda)$ we have to begin with the temporal difference algorithm $TD(\lambda)$. It has been shown that $TD(\lambda)$ converge to a good approximation of $V^{\rho}$ when a linear approximation is used (see Eq. (22)). At each step $t \in [0, T]$ of one trial $q \in [0, Q]$, we update the TD error as follow:

$$\begin{aligned} \delta_{t+1} &= \delta_t + \mathbf{z}_t(r_t + (\phi(\dot{\xi}^s_{t+1}) - \phi(\dot{\xi}^s_t))\mathbf{v}_q) \quad (35) \\ \mathbf{z}_{t+1} &= \lambda \mathbf{z}_t + \phi(\dot{\xi}^s_{t+1}), \quad (36) \end{aligned}$$

where $\delta_0 = \mathbf{0}$ and $\mathbf{z}_0 = \phi(\dot{\xi}^{\mathbf{s}}_{\mathbf{0}})$. $\mathbf{z}_t$ represents the eligibility trace. At the end of the trial, we update the weights $\mathbf{v}_q$ as follow:

$$\mathbf{v}_{q+1} = \mathbf{v}_q + \alpha_n \delta_T, \quad (37)$$

where $\alpha_n$ is a learning rate. By rearranging Eq. (36) and Eq. (37), we observe that the update of the weights $\mathbf{v}$ has the form:

$$\mathbf{v}_{q+1} = \mathbf{v}_q + \alpha_n(\mathbf{d} + \mathbf{C}\mathbf{v}_q + \mathbf{w}) \quad (38)$$

where

$$\mathbf{d} = E\{\textstyle\sum_{i=0}^{Q} \mathbf{z}_i \mathbf{r}_i\} \quad \mathbf{C} = E\{\textstyle\sum_{i=0}^{Q} \mathbf{z}_i(\phi(\dot{\xi}^s_{t+i}) - \phi(\dot{\xi}^s_i))\} \quad (39)$$

and $\mathbf{w}$ is a zero-mean noise. It has been shown in [3] that $\mathbf{v}$ converged to a fixed point $\mathbf{v}_\lambda$ satisfying $\mathbf{d} + \mathbf{C}\mathbf{v}_\lambda = 0$. However, $TD(\lambda)$ never explicitly compute $\mathbf{d}$ or $\mathbf{C}$.

The idea of the $LSTD(\lambda)$ algorithm is to explicitly compute from experience the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ as follow:

$$\mathbf{b} = \textstyle\sum_{i=0}^{t} \mathbf{z}_i \mathbf{r}_i \quad \mathbf{A} = \sum_{i=0}^{t} \mathbf{z}_i(\phi(\dot{\xi}^s_i) - \phi(\dot{\xi}^s_{i+1})) \quad (40)$$

Thus, after $n$ trials, $\mathbf{b}$ is an unbiased estimate of $n\mathbf{d}$ and $\mathbf{A}$ is an unbiased estimate of $-n\mathbf{C}$ and we are then able to estimate $\mathbf{v}_\lambda$ by computing $\mathbf{A}^{-1}\mathbf{b}$. The $LSTD - Q(\lambda)$ algorithm is the $LSTD(\lambda)$ applied to Eq. (26).