Imitation Learning with Non-Parametric Regression

Maarten Vaandrager, Robert Babuška, Lucian Buşoniu and Gabriel A.D. Lopes

***Abstract*—Humans are very fast learners. Yet, we rarely learn a task completely from scratch. Instead, we usually start with a rough approximation of the desired behavior and take the learning from there. In this paper, we use imitation to quickly generate a rough solution to a robotic task from demonstrations, supplied as a collection of state-space trajectories. Appropriate control actions needed to steer the system along the trajectories are then automatically learned in the form of a (nonlinear) state-feedback control law. The learning scheme has two components: a dynamic reference model and an adaptive inverse process model, both based on a data-driven, non-parametric method called local linear regression. The reference model infers the desired behavior from the demonstration trajectories, while the inverse process model provides the control actions to achieve this behavior and is improved online using learning. Experimental results with a pendulum swing-up problem and a robotic arm demonstrate the practical usefulness of this approach. The resulting learned dynamics are not limited to single trajectories, but capture instead the overall dynamics of the motion, making the proposed approach a promising step towards versatile learning machines such as future household robots, or robots for autonomous missions.**

I. Introduction

Most robots deployed to date are preprogrammed with controllers designed on the basis of accurate models and detailed task descriptions (e.g., industrial robots). This approach works well in the case that a predefined set of tasks have to be accomplished by the robot in a structured environment. Robots designed to operate in unstructured environments or under varying conditions typically rely on remote control by humans (e.g., space exploration robots). Surprisingly little progress has been achieved in the practical use of learning techniques. One of the reasons may be the fact that most learning control approaches attempt to learn entire control laws from scratch, assuming fairly general settings, such as in reinforcement learning [1]. Such algorithms typically require careful tuning of several parameters and need a long time to converge.

In this paper we take a different approach. Our premise is that a large class of robotic tasks can be demonstrated, either by a human, e.g., household tasks for future domestic robots, or by other 'teacher' robots that are more skilled than the learner — consider a group of robots reinforcing one another's skills in a soccer game. Thus we aim for robots that learn by imitating a teacher [2].

The importance of learning by imitation has been recognized in biology for quite some time now. Researchers such as Ramachandran [3] actually argue that imitation learning may have acted as the driving force behind the evolutionary leap of humans. Such strong statements together with the large body of work found in biology clearly motivate the use of imitation learning for programming robots.

In this paper we employ a technique termed *model-based imitation* (MBI), whose original idea stems from the work of Schaal [4], [5], [2]. However, while that work used reinforcement learning to find a policy that makes the system follow a demonstrated trajectory, in our work MBI yields a *reference model*. Given that model, the process is then controlled by an adaptive *inverse model* learned on-line from data. Local linear regression (LLR) [6] is used as a function approximation technique for both the reference and inverse models. Alternative ways to approximate the reference behavior include e.g. hidden Markov models and Gaussian mixture models [7], [8]; the behavior can also be represented using so-called motor primitives [9]. To follow the learned behavior, these works largely rely on classical control with an *a priori* model, whereas in our work the control uses the adaptive inverse model, learned at the same time as the reference model.

The learned dynamics obtained by coupling the reference and inverse models are not limited to single trajectories, but capture instead the overall dynamics of the motion. This makes our method fundamentally different from traditional trajectory tracking. For instance, if a robot following a prescribed trajectory collides with an unexpected obstacle, the trajectory-tracking control system, which has no notion of the obstacle, will attempt to push the robot through. Our controller, in contrast, will allow the robot to interrupt the motion and subsequently follow an alternative path from the collision state. This has been demonstrated in one of our experiments with a robotic arm. Another feature of our method should also be pointed out: once an inverse model has been learned while performing some demonstrated behavior, this inverse model is reusable for new behaviors, which are changed by replacing just the reference model.

In order to effectively use LLR in online learning for real systems, some nontrivial modifications to the basic algorithm are developed: a memory management mechanism that removes samples unneeded for an accurate representation of the function of interest, and a way to deal with noisy data.

After first introducing LLR in Section II, the online learning modifications to it are presented in Section III. Then, Section IV describes in more detail the control structure and overall algorithm for imitation learning. In Section V we illustrate imitation learning in two different real systems: an inverted pendulum and a robotic arm. Finally, Section VI concludes the paper.

M. Vaandrager is is with Plotprojects, Amsterdam, The Netherlands (e-mail: maarten@vaandrager.com). R. Babuška and G. Lopes are with the Delft Center for Systems and Control, Delft University of Technology, The Netherlands, e-mail: {r.babuska, g.a.d.lopes}@tudelft.nl. L. Buşoniu is with CNRS, Research Center for Automatic Control, University of Lorraine, Nancy, France and with the Department of Automation, Technical University of Cluj-Napoca, Romania (e-mail: lucian@busoniu.net).

II. Local Linear Regression

The goal of regression is to approximate an unknown function from input-output samples. Typically, given n input-output pairs of the form (x_i, y_i), an approximation $\hat{f}$ must be found that minimizes the loss:

$$\frac{1}{n}\sum_{i=1}^{n} \|y_i - \hat{f}(x_i)\|_2^2$$

where $\|\cdot\|_2$ denotes the Euclidean norm. In this paper we focus on a specific kind of sample-based representation of $\hat{f}$ denoted *local linear regression*[1]. This method is memory-based, also called "lazy" in the literature [10], [11] (in the sense that all computation is postponed until the function $\hat{f}$ must be evaluated), case-based, instance-based or experience-based.

LLR stores the input-output samples in a memory $\mathcal{M} = \{(x_i, y_i) \,|\, i = 1, \ldots, n\}$. When an approximation $\hat{f}$ must be computed at some query point x_q, the k samples in the memory closest to this point are found, in terms of some metric $\|\cdot\|$ on the input space. Then, a hyper-plane is fitted through the samples, and the predicted output $\hat{f}(x_q)$ is the value taken by this hyperplane at x_q. Thus, the method approximates nonlinear functions by piecewise affine functions.

Formally, the set of nearest-neighbor indices $\mathcal{N}_k(x_q)$ is defined by requiring it to satisfy the properties:

$$|\mathcal{N}_k(x_q)| = k$$
$$\|x_q - x_i\| \leq \|x_q - x_j\| \;\forall i \in \mathcal{N}_k(x_q), \forall j \notin \mathcal{N}_k(x_q)$$

(and, of course, to contain no duplicate indices). Relabel for convenience the memory samples so that the nearest neighbors are actually $1, \ldots, k$, and define also $\bar{x} = [x^T, 1]^T$, where the extra 1 will account for an affine term in the approximation. Then, two matrices X and Y are formed by putting together the inputs and outputs of the nearest neighbors:

$$X = \begin{bmatrix} \bar{x}_1 & \cdots & \bar{x}_k \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 & \cdots & y_k \end{bmatrix}$$

The local hyperplane is described by a vector β that satisfies $\beta^T X = Y$, which is typically an overdetermined system of equations because k is larger than the dimension of x. This system is solved, preferably with a numerically robust method such as Gaussian elimination, and with the resulting value of β we are ready to compute the function approximation for x_q: $\hat{f}(x_q, \mathcal{N}_k(x_q)) = \beta^T \bar{x}_q$. The nearest neighbors have been made explicit as an argument of $\hat{f}$ as this will help later in the paper.

LLR is summarized below (Algorithm 1) as a procedure that will be called as a component of the overall algorithm.

A crucial choice in the algorithm is the distance metric for the input space. In our studies a weighted L_1 (Manhattan) norm offered a good compromise between accuracy and computational cost: $\|x\| = \sum_d w_d |x_d|$, where d indexes the dimensions of x and the weights w_d are used to scale the inputs, bringing them into similar ranges. Therefore, this metric is used throughout our experiments.

[1]Note that the name *local linear regression* is typically used loosely in the literature to refer to *local affine regression*, which is the proper name for the method utilized in this paper.

Algorithm 1 LLR

Input: memory $\mathcal{M}$, parameter k, query point x_q
1: find k nearest neighbors $\mathcal{N}_k(x_q)$
2: construct X, Y, and solve linear system $\beta^T X = Y$
3: compute output $\hat{y}_q \leftarrow \beta^T \bar{x}_q$
Output: $\hat{y}_q$

The main drawback of LLR is the computational load of finding the k-nearest neighbors during the evaluation of $\hat{f}$. This can pose a real problem, since the number of samples required to accurately represent a function can at worst grow exponentially with the dimension of the input space. However, we have found that given today's available computational power, this method can be used efficiently in relatively high dimensional spaces, such as the robot manipulator arm example presented in Section V-B.

III. Online learning in LLR

Online learning is necessary to adapt the inverse model in MBI. In its most basic form, learning in LLR would be performed by simply adding new samples to the database. However, storing all observations throughout the system's operational life would require an indefinitely large memory. Moreover, the model function being approximated can be time-varying, making old observations obsolete. This motivates removing judiciously chosen samples to keep the database relevant and within a reasonable size. Furthermore, in real systems observations are corrupted by noise, and its effects must be controlled. In this section we introduce heuristics to deal with both of these issues.

A. Memory management

In LLR, only the nonlinear parts of the function need to be densely populated with samples, while the method naturally interpolates the linear parts with only a few samples (on the order of k). Thus we propose to adapt the database (memory) in order to span the nonlinear parts with more samples, providing a variable sample distribution and a variable size of the local neighborhood. This approach is similar to the so-called condensed nearest neighbor classification [12], which tries to find and retain only those samples that lie on the decision boundaries between classification categories [13]. This results in a piecewise constant approximation of the function with values between the decision boundaries being interpolated as constant. Here we adapt this idea to the context of regression.

Specifically, after a maximum size of the memory is reached, every new sample replaces an older sample that best fitted the local linear models for past query points. To implement this, we attach to every sample (x_i, y_i) an additional scalar value ϵ_i representing relevance, defined as the running average of the distance between the outputs y_i and the local models $\hat{f}(x_i, \mathcal{N}_k(x_q))$. So the memory $\mathcal{M}$ is

now composed of samples having the form (x_i, y_i, ϵ_i). Every time a sample i is used in estimating a model for the input x_q (i.e. when $i \in \mathcal{N}_k(x_q)$), its relevance value is incrementally updated with the new difference from the local model:

$$\epsilon_i \leftarrow \gamma\epsilon_i + (1-\gamma)\|y_i - \hat{f}(x_i, \mathcal{N}_k(x_q))\|_2^2$$

where γ is a parameter in $(0, 1]$. Note that the running average ϵ_i is only updated at those instants when the sample is used in regression. When ϵ_i has a low value, it means that the function is (almost) linear in the sample's neighborhood, so the samples are dense enough in that neighborhood. Conversely, when ϵ_i has a large value the neighborhood is nonlinear. Thus, replacing samples with the lowest relevance ϵ_i removes samples at linear parts and preserves samples in nonlinear parts, finally resulting in an approximation of the function with a variable sample distribution but a uniform approximation error.

The process of replacing samples on the basis of relevance is referred to as *memory management* in the sequel. Memory management allows us to continually insert new information and discard redundant information.

B. Noisy observations

Noise is intrinsic to any real system, and a strategy to deal with it should be devised. A distinction between noisy observations and informational observations should be made, however this is a very difficult or even impossible task in general [14]. Here, we adopt a pragmatic solution based on the observation that the local least squares solution is the best average of the first order relation present in the nearest neighbors. Thus, by adjusting the outputs of these neighbors to fit the model one can effectively reduce noise. Formally, for every query point x_q, we set:

$$y_i \leftarrow \hat{f}(x_i, \mathcal{N}_k(x_q)), \quad \forall i \in \mathcal{N}_k(x_q)$$

Note the neighbors of x_q are used and not those of x_i.

C. Learning LLR algorithm and illustration

Algorithm 2 programmatically describes the online learning component of LLR, with memory management and sample adjustment.

Algorithm 2 LLR_Learn

Input: $\mathcal{M}$, k, γ, $n_{\max}$, new sample (x_q, y_q)

1: find $\mathcal{N}_k(x_q)$, construct X, Y, and solve $\beta^T X = Y$
2: initialize sample relevance: $\epsilon \leftarrow \|y_q - \hat{f}(x_q, \mathcal{N}_k(x_q))\|_2^2$
3: add new sample: $\mathcal{M} \leftarrow \mathcal{M} \cup (x_q, y_q, \epsilon)$
4: **for all** $i \in \mathcal{N}_k(x_q)$ **do**
5: update relevance:
$\epsilon_i \leftarrow \gamma\epsilon_i + (1-\gamma)\|y_i - \hat{f}(x_i, \mathcal{N}_k(x_q))\|_2^2$
6: adjust output to local model: $y_i \leftarrow \hat{f}(x_i, \mathcal{N}_k(x_q))$
7: **end for**
8: if $|\mathcal{M}| > n_{\max}$, remove least relevant sample:
$\mathcal{M} \leftarrow \mathcal{M} \setminus \{(x_j, y_j, \epsilon_j)\}, j = \arg\min_i(\epsilon_i)$

Output: $\mathcal{M}$

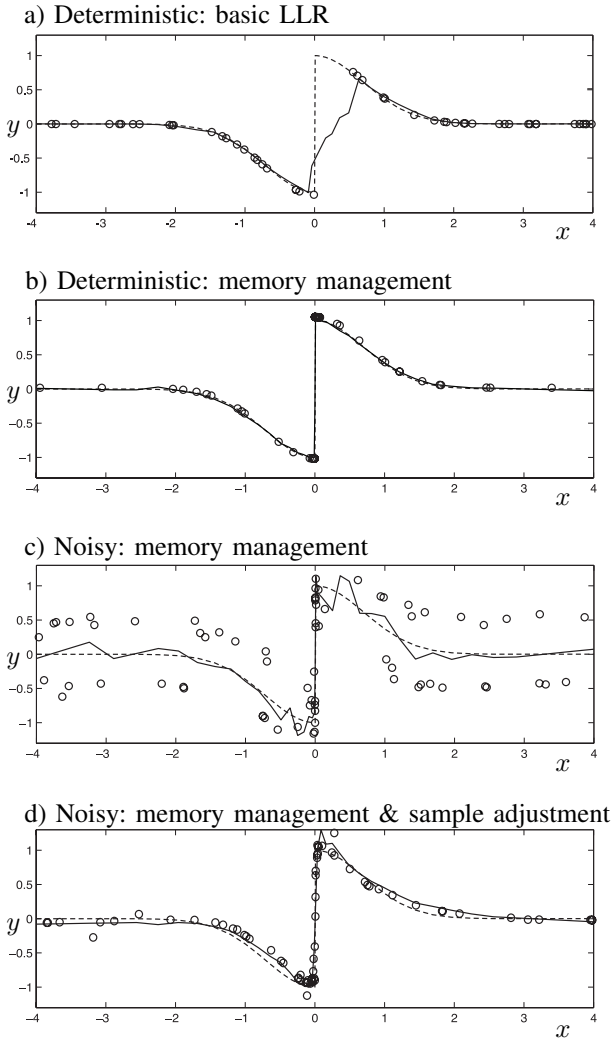

Fig. 1. Illustration of local linear regression for deterministic and noisy samples. Dashed lines represent the real function f, solid lines represent the approximated function $\hat{f}$ by LLR, and circles represent the samples.

Figure 1 illustrates the effect of memory management and, separately, sample adjustment. Specifically, Figure 1(a) and (b) shows results with deterministic observations, respectively with and without memory management (sample adjustment is not used). Memory management has a positive effect on the overall approximation of the nonlinear function, as the LLR fits the true function nearly perfectly after observing and deleting 1000 samples. Figure 1(c) and (d) shows results with noisy observations, with and without sample adjustment (memory management is used). Sample adjustment improves prediction accuracy, but since one keeps integrating noisy samples in a memory of a limited size, a perfect approximation can never be reached.

IV. Imitation learning

Imitation learning controls the system with an adaptive, nonlinear state feedback obtained as the composition of two elements: the reference model and the inverse model, as

shown in Figure 2 (in this figure, t denotes the discrete time step). The reference model infers the desired behavior from demonstrations, while the inverse model is used as a controller to follow the reference trajectories. Denoting the reference model by $R : S \to S$ where S is the state space, it produces for any state s a desired next state $\hat{s}' = R(s)$. For robotic systems, the state typically contains positions and velocities. The inverse model, denoted by $\hat{P}^{-1} : S \times S \to U$, maps a desired transition $(s, \hat{s}')$ into a control input that will (approximately) realize this transition, $\hat{u} = \hat{P}^{-1}(s, \hat{s}')$.

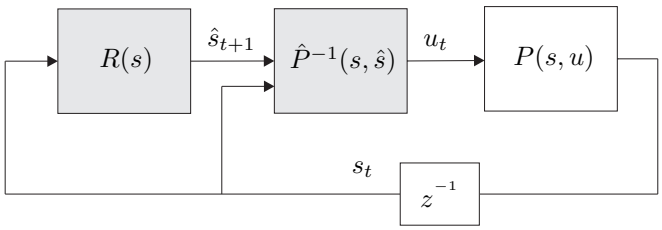

Fig. 2. Model-based imitation scheme.

The reference model R is approximated by applying LLR on demonstrated state trajectories, collected together in a database of input-output samples of the form $(x = s_t, y = s_{t+1})$. The inverse process model $\hat{P}^{-1}$ is learned with the online variant of LLR, from observations of the process state and control action, stored as input-output samples of the form $(x = [s_t^T, s_{t+1}^T]^T, y = u_t)$. The entire procedure is summarized as Algorithm 3, where subscripts R and P are used to differentiate entities associated with the reference and process model, respectively.

Algorithm 3 Imitation learning

Input: $\mathcal{M}_R, k_R, k_P, \gamma_P$

1: $\mathcal{M}_P \leftarrow \emptyset$ (or existing model, if available)
2: **for** each step $t = 0, 1, 2, \ldots$ **do**
3: measure state s_t
4: $\hat{s}_{t+1} = \text{LLR}(\mathcal{M}_R, k_R, s_t)$
5: $\hat{u}_t = \text{LLR}(\mathcal{M}_P, k_P, [s_t^T, \hat{s}_{t+1}^T]^T)$
6: apply $\hat{u}_t$ to system
7: if $t \geq 1$, learn from previous transition
8: $\mathcal{M}_P \leftarrow \text{LLR\_Learn}(\mathcal{M}_P, k_P, \gamma_P, [s_{t-1}^T, s_t^T]^T, \hat{u}_{t-1})$
9: **end for**

Note the learning step 8 at time t is performed before measuring the next state s_{t+1}, so it has to work with the previous transition. This is because waiting to measure s_{t+1} and then performing learning would introduce too large delays in the real-time control (i.e. before applying the corresponding action $\hat{u}_{t+1}$).

V. Examples

A. Pendulum Swing-up

As a first experiment, we used the inverted pendulum swing-up problem to study the performance of imitation learning. The inverted pendulum is realized by placing an off-center weight on a vertical disk driven by a DC motor, as shown in Figure 3. The goal is to bring the weight from any initial state to the upper unstable equilibrium and stabilize it there. However, the control voltage is constrained such that it is insufficient to push the pendulum up in a single rotation from every initial state. The pendulum needs to be swung back and forth (destabilized) to gather energy, prior to being pushed up and stabilized.

The states s are the angle $\alpha \in [-\pi, \pi)$ rad, with $\alpha = 0$ pointing up, and the angular velocity $\dot{\alpha}$. The control input u is a voltage is constrained to $[-3, 3]$ V. The goal is to stabilize the pendulum in the unstable equilibrium $x = 0$ (pointing up). The controller is implemented in Matlab, using the sampling period $T_\text{s} = 0.03$ s. Note that a model of this system is:

$$\ddot{\alpha} = \frac{1}{J}\left(mgl\sin(\alpha) - b\dot{\alpha} - \frac{K^2}{R}\dot{\alpha} + \frac{K}{R}u\right)$$

with $J = 1.91 \cdot 10^{-4}\,\text{kgm}^2$, $m = 0.055\,\text{kg}$, $g = 9.81\,\text{m/s}^2$, $l = 0.042\,\text{m}$, $b = 3 \cdot 10^{-6}\,\text{kg/s}$, $K = 0.0536\,\text{Nm/A}$, $R = 9.5\,\Omega$, but the learning controller does not use this information. The only prior knowledge about the system entering the control algorithm is the structure of the state and control variables.

In our experiments, we first demonstrated how the controller should swing the pendulum up and then applied imitation learning to train the inverse-model controller. Ten swing-ups (5 clockwise and 5 counterclockwise) were demonstrated by turning the disk by hand, as illustrated in Figure 3. No control input was generated, so the only information recorded are the state variables. The corresponding phase-plane trajectories are shown in Figure 4.

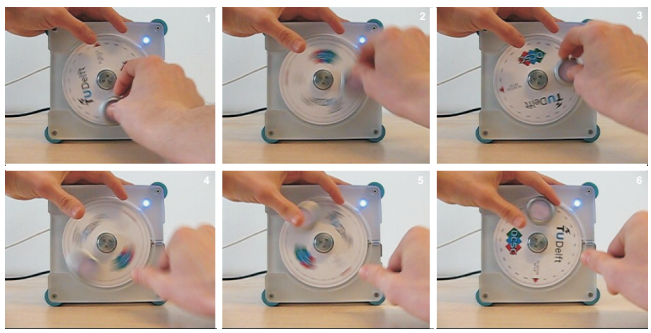

Fig. 3. Demonstration of pendulum swing up by hand.

By applying LLR to the demonstrated trajectories, we obtain the reference model which in every state calculates the desired next state. Each sample in the reference model R is defined by $([\alpha_t, \dot{\alpha}_t]^T, [\alpha_{t+1}, \dot{\alpha}_{t+1}]^T)$. The number k of nearest neighbors was 10 and the size of the entire memory was 660 samples. The inverse process model $\hat{P}^{-1}$ stores samples $([\alpha_t, \dot{\alpha}_t, \alpha_{t+1}, \dot{\alpha}_{t+1}]^T, u_t)$. For this approximator, the number of neighbors was $k = 15$, the memory size was 1000 samples,

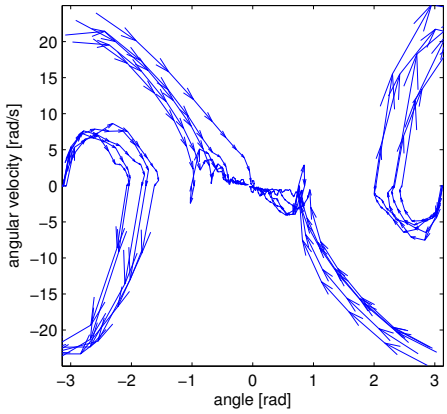

Fig. 4. Demonstrated reference model R for the inverted pendulum.

and the parameter $\gamma = 0.9$ was used in memory management. The resulting control policy is successful in imitating the demonstrated behavior and thus swinging up the pendulum, as illustrated in Figure 5. Figure 6 shows that the process state follows the desired state and that the actuation is quite smooth.[2]

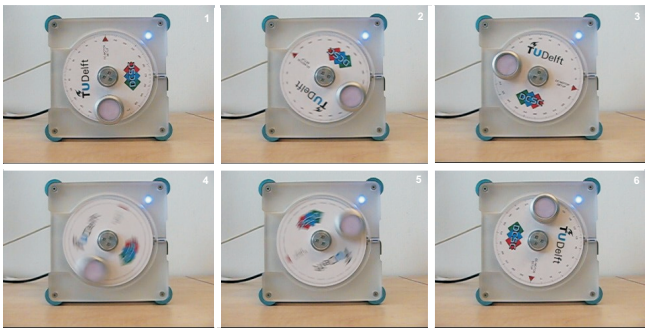

Fig. 5. Imitation of the demonstrated behavior.

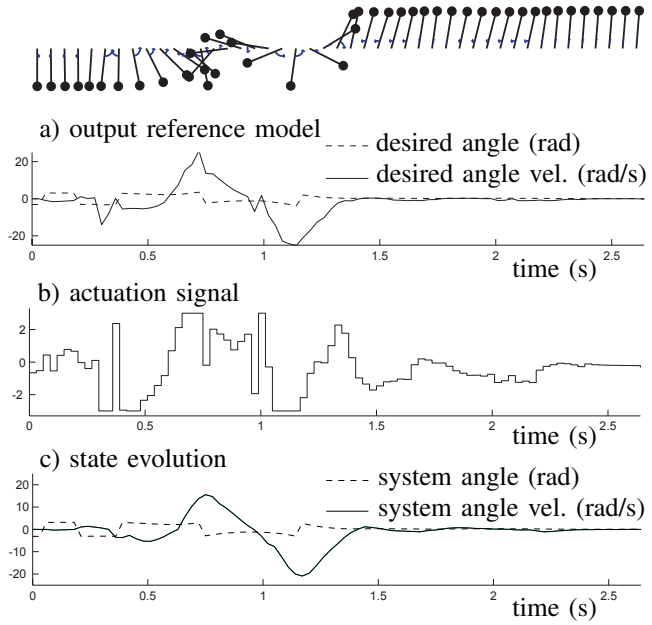

Fig. 6. Evolution of the desired state $\hat{s}$, resulting actuation u and the state evolution s by model-based imitation.

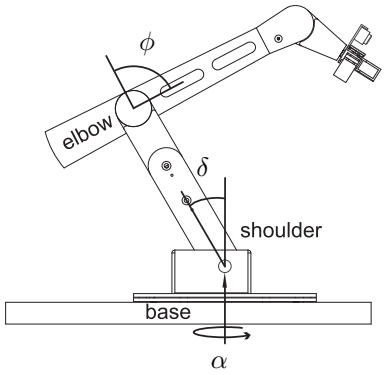

Fig. 7. EdRo — educational robotic manipulator. Three joints were used in the imitation, the base joint α, the shoulder joint δ, and the elbow joint ϕ.

B. Robotic Manipulator

In this example we use a more complicated system – a robotic manipulator Ed-Ro, which is a lightweight, low-cost robot developed mainly for educational purposes. It has five revolute joints and a two-fingered gripper, all actuated by DC motors. The manipulator is controlled from Matlab via an RS232 or USB serial port interface. In this experiment we learn the control of three joints: the base (α), the shoulder (δ) and the elbow (ϕ), see Figure 7. Therefore, the reference model R has samples of the form $([\alpha_t, \dot{\alpha}_t, \delta_t, \dot{\delta}_t, \phi_t, \dot{\phi}_t]^T, [\alpha_{t+1}, \dot{\alpha}_{t+1}, \delta_{t+1}, \dot{\delta}_{t+1}, \phi_{t+1}, \dot{\phi}_{t+1}]^T)$ and the inverse model $\hat{P}^{-1}$ of the form $([\alpha_t, \dot{\alpha}_t, \delta_t, \dot{\delta}_t, \phi_t, \dot{\phi}_t, \alpha_{t+1}, \dot{\alpha}_{t+1}, \delta_{t+1}, \dot{\delta}_{t+1}, \phi_{t+1}, \dot{\phi}_{t+1}]^T, [u_{\alpha,t}, u_{\delta,t}, u_{\phi,t}]^T)$.

As an example of the desired behavior we considered a path to be followed by the end-effector in a 3D space. The behavior was demonstrated by manually moving the end-effector along the path (approximately three times during about 10 seconds). Several snapshots of the demonstration are shown in Figure 8.

The state trajectories were recorded with a sample period of $T_s = 0.05$, yielding about 200 samples, the memory size of the reference model. During imitation, $k = 15$ nearest neighbors are used for the reference model and $k = 40$ for the inverse plant model, which has a memory size of 1000 samples. The value of parameter γ is 0.95. Figure 9 illustrates snapshots of the behavior controlled by imitation, while Figure 10 shows sample trajectories for demonstration (a) and imitation after learning the inverse model (b). The two sets of trajectories are qualitatively equivalent.

[2]Videos for these experiments are available at busoniu.net/projects.php, see project "Using prior knowledge to accelerate reinforcement learning".

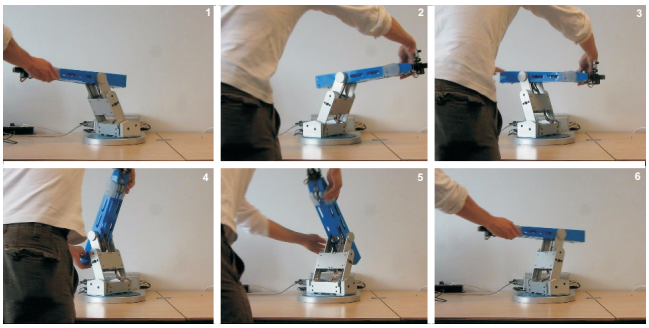

Fig. 8. The desired behavior is demonstrated by moving the robotic manipulator by hand.

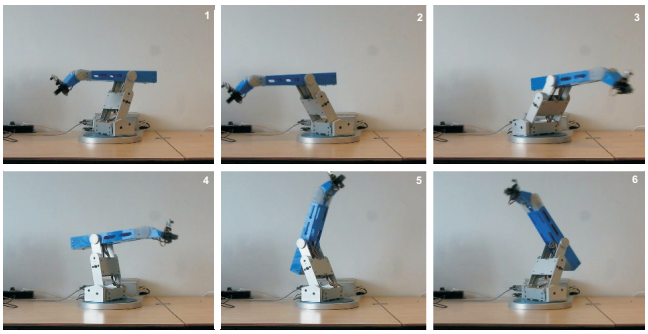

Fig. 9. Imitation of the demonstrated behavior.

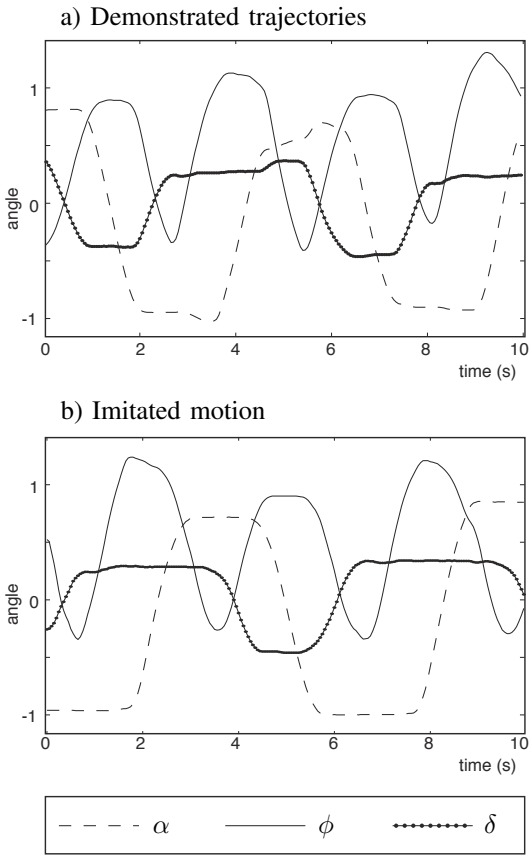

Fig. 10. Sample state trajectories for the EdRo robotic manipulator during demonstration and imitation.

Since the MBI scheme presented in this paper imitates the reference dynamics of the demonstration and not the exact trajectories, we found that when presenting a circular motion to the end effector, the robot would move in an arbitrary size circular pattern. However, by preventing the LLR algorithm to extrapolate from its data set, the motion of the robot always stays within the domain of the demonstrations, and remains safe.[3]

VI. Conclusions

In this paper we have proposed and demonstrated a simple, but very effective method for robot learning by imitation. By taking advantage of nonparametric function approximation methods such as local linear regression, we showed that it is possible to both imitate a demonstrated motion and approximate the plant model simultaneously in real-time.

We are now investigating the idea of employing model-based imitation in reinforcement learning in order to speed up the convergence of the learning controller.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[2] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in Cognitive Sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[3] V. S. Ramachandran and D. Rogers, "It's all done with mirrors," *Scientific American Mind*, vol. 18, no. 4, pp. 16–18, 2007.

[3] As for the previous experiments, see busoniu.net/projects.php for videos illustrating the learning process and the real-time control performance.

[4] S. Schaal, "Learning from demonstration," in *Advances in Neural Information Processing Systems 9*, M. Mozer, M. I. Jordan, and T. Petsche, Eds. MIT Press, 1997, pp. 1040–1046.

[5] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *Proceedings 14th International Conference on Machine Learning (ICML-97)*, Nashville, US, 8–12 July 1997, pp. 12–20.

[6] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, vol. 11, no. 1–5, pp. 11–73, 1997.

[7] S. Calinon, F. D'Halluin, E. Sauser, D. Caldwell, and A. Billard, "Learning and reproduction of gestures by imitation," *IEEE Robotics and Automation Magazine*, vol. 17, no. 2, pp. 44–54, 2010.

[8] S. S.M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with gaussian mixture models," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 943–957, 2011.

[9] J. Kober and J. Peters, "Imitation and reinforcement learning," *IEEE Robotics and Automation Magazine*, vol. 17, no. 2, pp. 55–62, 2010.

[10] T. M. Dietrich Wetterschereck, David Aha, "A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms," *Artificial Intelligence Review*, vol. 11, pp. 273–314, 1997.

[11] G. Bontempi, M. Birattari, and H. Bersini, "Lazy learning for modeling and control design," *International Journal of Control*, vol. 72, pp. 643–658, 1999.

[12] P. Tsaparas, "Nearest neighbor search in multidimensional spaces," Department of Computer Science, University of Toronto, Tech. Rep. 319-02, 1999.

[13] D. Barschdorff, A. Bothe, U. Gärtner, and A. Jäger, "Retraining and redundancy elimination for a condensed nearest neighbour network," in *Proceedings 5th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-92)*. Paderborn, Germany: Springer, 9–12 June 1992, pp. 246–255.

[14] D. R. Wilson and T. R. Martinez, "Reduction techniques for exemplar-based learning algorithms," *Machine Learning*, vol. 38, no. 3, pp. 257–286, 2000.