

Technical report 10-064

# **Exploiting Policy Knowledge in Online Least-Squares Policy Iteration: An Empirical Study\***

L. Buşoniu, B. De Schutter, R. Babuška, and D. Ernst

*To cite this work, please refer to the published version:*

L. Buşoniu, B. De Schutter, R. Babuška, and D. Ernst, “Exploiting policy knowledge in online least-squares policy iteration: An empirical study,” *Automation, Computers, Applied Mathematics*, vol. 19, no. 4, pp. 521–529, 2010.

Delft Center for Systems and Control  
Delft University of Technology  
Mekelweg 2, 2628 CD Delft  
The Netherlands  
phone: +31-15-278.24.73 (secretary)  
URL: <https://www.dcsc.tudelft.nl>

---

\* This report can also be downloaded via <https://dpub.eu/10-064>

# Exploiting policy knowledge in online least-squares policy iteration: An empirical study

Lucian Buşoniu  
Team SequEL  
INRIA Lille – Nord Europe  
France  
lucian@busoniu.net

Bart De Schutter, Robert Babuška  
Delft Center for Systems and Control  
Delft University of Technology  
The Netherlands  
{b.deschutter, r.babuska}@tudelft.nl

Damien Ernst  
Institut Montefiore  
FNRS & University of Liège  
Belgium  
dernst@ulg.ac.be

**Abstract**—Reinforcement learning (RL) is a promising paradigm for learning optimal control. Traditional RL works for discrete variables only, so to deal with the continuous variables appearing in control problems, approximate representations of the solution are necessary. The field of approximate RL has tremendously expanded over the last decade, and a wide array of effective algorithms is now available. However, RL is generally envisioned as working without any prior knowledge about the system or the solution, whereas such knowledge is often available and can be exploited to great advantage. Therefore, in this paper we describe a method that exploits prior knowledge to accelerate online least-squares policy iteration (LSPI), a state-of-the-art algorithm for approximate RL. We focus on prior knowledge about the monotonicity of the control policy with respect to the system states. Such monotonic policies are appropriate for important classes of systems appearing in control applications, including for instance nearly linear systems and linear systems with monotonic input nonlinearities. In an empirical evaluation, online LSPI with prior knowledge is shown to learn much faster and more reliably than the original online LSPI.

**Index Terms**—reinforcement learning, prior knowledge, least-squares policy iteration, online learning.

## I. INTRODUCTION

Reinforcement learning (RL) can address problems from a variety of fields, including automatic control, computer science, operations research, and economics [1]–[4]. In automatic control, RL algorithms can solve nonlinear, stochastic optimal control problems, in which a cumulative reward signals must be maximized. Rather than using a model as in classical control, a RL controller learns how to control the system from data, and online RL techniques collect their own data by interacting with the system. For systems with continuous or large discrete state-action spaces, as most of the systems encountered in automatic control, RL solutions cannot be represented exactly, but must be approximated [4]. State-of-the-art algorithms for approximate RL use weighted summations of basis functions to represent the value function (which gives the cumulative reward as a function of the states and possibly of the actions), and least-squares techniques to find the weights. Such techniques have been studied by many authors [5]–[10], and recently surveyed in e.g. [11]–[13].

One such algorithm is least-squares policy iteration (LSPI) [7]. At every iteration, LSPI evaluates the current control

policy, by computing its approximate value function from transition samples, and then finds a new, improved policy from this value function. LSPI can efficiently use transition data collected in any manner, but originally works offline. In [14], we have introduced an online variant of LSPI, which collects its own data by interacting with the system, and performs policy improvements “optimistically” [3], [15], without waiting until an accurate evaluation of the current policy is completed. Such policy improvements allow online LSPI to learn fast, i.e., to achieve good performance after interacting with the system for only a short interval of time.

In this paper, we present a method to exploit prior knowledge in order to improve the learning speed of online LSPI. Although RL is usually envisioned as working without any prior knowledge, such knowledge is often available, and exploiting it can be highly beneficial. We consider prior knowledge in the form of the monotonicity of the control policy with respect to the state variables. Such monotonic policies are suitable for controlling important classes of systems. For instance, policies that are linear in the state variables, and therefore monotonic, work well for controlling linear systems, as well as nonlinear systems in neighborhoods of equilibria where they are nearly linear. Moreover, some linear systems with monotonic input nonlinearities (such as saturation or dead-zone nonlinearities) have policies that, while strongly nonlinear, are nevertheless monotonic.

We employ a policy representation for which monotonicity can be ensured by imposing linear inequality constraints on the policy parameters. This allows policy improvements to be performed efficiently, using quadratic programming. A speedup of the learning process is expected, because online LSPI restricts its focus to the class of monotonic policies, and no longer invests valuable learning time in trying other, unsuitable policies. The effects of using prior knowledge in online LSPI are illustrated in a simulation study involving the stabilization of a DC motor.

Several other online RL algorithms based on policy iteration and least-squares techniques have been proposed. For instance, [16] investigated a version of LSPI with online sample collection, focusing on the issue of exploration. This version does not perform optimistic policy updates, but fully executes offline LSPI between consecutive sample-collection episodes. An algorithm related to LSPI, called least-squares policy evaluation [6], was studied in the optimistic context

Based on the paper “Using prior knowledge to accelerate least-squares policy iteration” by Buşoniu, De Schutter, Babuška, and Ernst, which appeared in *Proceedings of the 2010 IEEE International Conference on Automation Quality and Testing Robotics (AQTR)*, © 2010 IEEE.

in [17]. Another optimistic variant of policy iteration based on least-squares methods was given by [10]. However, these techniques do not exploit prior knowledge about the solution.

The remainder of this paper is organized as follows. The necessary theoretical background on RL, together with offline and online LSPI, are described in Section II. Then, in Section III, we introduce our method to integrate prior knowledge into online LSPI, and in Section IV, we present the results of our simulation experiments. Section V concludes the paper and outlines some promising directions for future research.

Note that this article is a revised and extended version of [18]. New, more detailed experimental results and insights are provided, while the introduction of the various algorithms has been extended.

## II. PRELIMINARIES: REINFORCEMENT LEARNING AND ONLINE LSPI

This section first summarizes the RL problem in the framework of Markov decision processes, following [4]. Then, offline LSPI [7] and the online LSPI algorithm we introduced in [14] are described.

### A. Markov decision processes and classical policy iteration

Consider a Markov decision process with state space  $X$  and action space  $U$ . Assume for now that  $X$  and  $U$  are countable. The probability that the next state  $x_{k+1}$  is reached after action  $u_k$  is taken in state  $x_k$  is  $f(x_k, u_k, x_{k+1})$ , where  $f: X \times U \times X \rightarrow [0, 1]$  is the transition probability function. After the transition to  $x_{k+1}$ , a reward  $r_{k+1} = \rho(x_k, u_k, x_{k+1})$  is received, where  $\rho: X \times U \times X \rightarrow \mathbb{R}$  is the reward function. The symbol  $k$  denotes the discrete time index. The expected infinite-horizon discounted return of initial state  $x_0$  under a control policy  $h: X \rightarrow U$  is:

$$R^h(x_0) = \lim_{K \rightarrow \infty} \mathbb{E}_{x_{k+1} \sim f(x_k, h(x_k), \cdot)} \left\{ \sum_{k=0}^K \gamma^k r_{k+1} \right\} \quad (1)$$

where  $r_{k+1} = \rho(x_k, u_k, x_{k+1})$ ,  $\gamma \in [0, 1]$  is the discount factor, and the notation  $x_{k+1} \sim f(x_k, h(x_k), \cdot)$  means that  $x_{k+1}$  is drawn from the distribution  $f(x_k, h(x_k), \cdot)$ . The goal is to find an optimal policy  $h^*$  that maximizes the return (1) from every  $x_0 \in X$ . RL algorithms aim to find  $h^*$  from transition and reward data, without using the functions  $f$  and  $\rho$ . Moreover, *online* RL algorithms do not even require data in advance, but collect their own data, by interacting with the system while they learn.

The classical policy iteration algorithm starts with some initial policy  $h_0$ . At every iteration  $\tau \geq 0$ , the algorithm first *evaluates* the current policy  $h_\tau$  by computing its Q-function  $Q_\tau: X \times U \rightarrow \mathbb{R}$ , which gives for every pair  $(x, u)$  the expected return when starting in  $x$ , applying  $u$ , and following  $h_\tau$  thereafter. This Q-function is the unique solution of the Bellman equation:

$$Q_\tau = T_\tau(Q_\tau) \quad (2)$$

where the Bellman mapping  $T_\tau$  is:

$$[T_\tau(Q)](x, u) = \mathbb{E}_{x' \sim f(x, u, \cdot)} \{ \rho(x, u, x') + \gamma Q(x', h_\tau(x')) \}$$

Once  $Q_\tau$  is available, an *improved* policy is determined:

$$h_{\tau+1}(x) \leftarrow \arg \max_u Q_\tau(x, u) \quad (3)$$

and the algorithm continues with this policy at the next iteration. Policy iteration is guaranteed to converge to  $h^*$ , see, e.g., Section 4.3 of [2].

### B. Least-squares policy iteration

When the state-action space is very large, Q-functions cannot be represented exactly, but must be approximated. Here, linearly parameterized approximators are considered:

$$\widehat{Q}(x, u) = \phi^\top(x, u) \theta \quad (4)$$

where  $\phi(x, u)$  is a vector of  $n$  basis functions (BFs),  $\phi(x, u) = [\phi_1(x, u), \dots, \phi_n(x, u)]^\top$ , and  $\theta \in \mathbb{R}^n$  is a parameter vector. Given this approximator, the policy evaluation problem at the  $\tau$ th iteration, for the policy  $h_\tau$ , boils down to finding  $\theta_\tau$  so that  $\widehat{Q}_\tau \approx Q_\tau$ , where  $Q_\tau(x, u) = \phi^\top(x, u) \theta_\tau$ .

LSPI [7] is an originally offline RL algorithm that finds  $\widehat{Q}_\tau$  by solving a *projected* form of the Bellman equation:

$$\widehat{Q}_\tau = P_w(T_\tau(\widehat{Q}_\tau)) \quad (5)$$

where  $P_w$  performs a weighted least-squares projection onto the space of representable Q-functions, i.e., the space  $\{ \phi^\top(x, u) \theta \mid \theta \in \mathbb{R}^n \}$  spanned by the BF's. The weight function  $w: X \times U \rightarrow [0, 1]$  must satisfy  $\sum_{x, u} w(x, u) = 1$ , because it is also interpreted as a probability distribution. Note that the algorithm is called “least-squares” because (5) is, in a sense, a least-squares approximation of the original Bellman equation (2).

Equation (5) is rewritten as a linear system in the parameters:

$$\Gamma \theta_\tau = z$$

where the matrix  $\Gamma \in \mathbb{R}^{n_s \times n}$  and the vector  $z \in \mathbb{R}^{n_s}$  can be estimated from samples (note  $\Gamma$  and  $z$  are different at every iteration, since they depend on the policy). More specifically, consider a set of  $n_s$  samples  $\{(x_{l_s}, u_{l_s}, x'_{l_s} \sim f(x_{l_s}, u_{l_s}, \cdot), r_{l_s} = \rho(x_{l_s}, u_{l_s}, x'_{l_s})) \mid l_s = 1, \dots, n_s\}$ , where the probability of each pair  $(x, u)$  is  $w(x, u)$ . The estimates  $\widehat{\Gamma}$  and  $\widehat{z}$  are initialized to zeros and updated for every sample  $l_s = 1, \dots, n_s$  as follows:

$$\begin{aligned} \widehat{\Gamma} &\leftarrow \widehat{\Gamma} + \phi(x_{l_s}, u_{l_s}) \phi^\top(x_{l_s}, u_{l_s}) - \gamma \phi(x_{l_s}, u_{l_s}) \phi^\top(x'_{l_s}, h_\tau(x'_{l_s})) \\ \widehat{z} &\leftarrow \widehat{z} + \phi(x_{l_s}, u_{l_s}) r_{l_s} \end{aligned}$$

After processing the entire batch of samples, an estimated parameter vector  $\widehat{\theta}_\tau$  is found by solving:

$$\frac{1}{n_s} \widehat{\Gamma} \widehat{\theta}_\tau = \frac{1}{n_s} \widehat{z}$$

This equation can be solved in several ways, e.g., (i) by matrix inversion, (ii) by Gaussian elimination, or (iii) by incrementally computing the inverse with the Sherman-Morrison formula. When  $n_s \rightarrow \infty$  and under appropriate conditions,  $\frac{1}{n_s} \widehat{\Gamma} \rightarrow \Gamma$  and  $\frac{1}{n_s} \widehat{z} \rightarrow z$ , and therefore  $\widehat{\theta}_\tau \rightarrow \theta_\tau$  (see [7], Chapter 6 of [3]). Note that in the sequel, we drop the hat notation

for  $\Gamma$ ,  $z$ , and  $\theta_\tau$ , with the implicit understanding that they are always estimates.

Once a parameter vector is available, it is used in (4) to obtain an approximate Q-function, thus completing the *policy evaluation* step. In turn, this approximate Q-function is plugged into (3) to perform the *policy improvement* step, and the LSPI algorithm continues with the improved policy at the next iteration.

LSPI is data-efficient and, as long as the policy evaluation error is bounded, eventually produces policies with a bounded suboptimality. Note that although for the derivation above we assumed that  $X$  and  $U$  are countable, LSPI can also be applied in uncountable spaces, such as the continuous spaces found in most automatic control problems. For a more detailed description of LSPI, see [7].

### C. Online LSPI

In this paper, we consider an *online* variant of LSPI, which collects its own transition samples by interacting with the system [14]. This online variant is shown in Algorithm 1. Note that an idealized, infinite-time setting is considered, in which the algorithm runs forever and its result is the performance improvement achieved while interacting with the system. In practice, the algorithm is of course stopped after a finite time.

---

#### Algorithm 1 Online LSPI with $\varepsilon$ -greedy exploration.

---

**Input:** BFs  $\phi_1, \dots, \phi_n$ ;  $\gamma$ ;  $K_\theta$ ;  $\{\varepsilon_k\}_{k \geq 0}$ ;  $\delta$

- 1:  $\tau \leftarrow 0$ ; initialize policy  $h_0$
- 2:  $\Gamma \leftarrow \delta I_{n \times n}$ ;  $z \leftarrow 0_n$
- 3: measure initial state  $x_0$
- 4: **for** each time step  $k \geq 0$  **do**
- 5:  $u_k \leftarrow \begin{cases} h_\tau(x_k) & \text{w.p. } 1 - \varepsilon_k \\ \text{a uniform random action} & \text{w.p. } \varepsilon_k \end{cases}$
- 6: apply  $u_k$ , measure next state  $x_{k+1}$  and reward  $r_{k+1}$
- 7:  $\Gamma \leftarrow \Gamma + \phi(x_k, u_k) \phi^\top(x_k, u_k)$   
 $\quad - \gamma \phi(x_k, u_k) \phi^\top(x_{k+1}, h_\tau(x_{k+1}))$
- 8:  $z \leftarrow z + \phi(x_k, u_k) r_{k+1}$
- 9: **if**  $k = (\tau + 1)K_\theta$  **then**
- 10: find  $\theta_\tau$  by solving  $\frac{1}{k+1} \Gamma \theta_\tau = \frac{1}{k+1} z$
- 11:  $h_{\tau+1}(x) \leftarrow \arg \max_u \phi^\top(x, u) \theta_\tau$
- 12:  $\tau \leftarrow \tau + 1$
- 13: **end if**
- 14: **end for**

---

To achieve fast learning, online LSPI performs policy improvements once every few transitions, without waiting until an accurate evaluation of the current policy is completed (unlike the offline algorithm, which processes all the samples at every iteration, to obtain an accurate policy evaluation). The integer  $K_\theta \geq 1$  is the number of transitions between two consecutive policy improvements. Such a variant of policy iteration is called “optimistic” [3], [15] (it is “fully optimistic” if a policy improvements is performed after each transition, and “partially optimistic” otherwise).

Online LSPI must also explore, i.e., try other actions than those given by the current policy. Exploration helps to collect informative data about (i) the performance of actions different from those taken by the current policy, and (ii) regions of the state space that would not be reached using this policy. In Algorithm 1,  $\varepsilon$ -greedy exploration is employed, which applies at every step  $k$  a uniformly random exploratory action with probability  $\varepsilon_k \in [0, 1]$ , and the action indicated by the policy with probability  $1 - \varepsilon_k$ , see, e.g., Section 2.2 of [2]. Typically,  $\varepsilon_k$  decreases over time, so that the algorithm increasingly exploits the current policy. Furthermore, to ensure the invertibility of  $\Gamma$  in the early stages of the learning process, this matrix is initialized to a small multiple of the identity matrix, using the parameter  $\delta > 0$ .

Note that, in practice, online LSPI does not have to compute and store *complete* improved policies (line 11). Indeed, such a procedure would be problematic in large and continuous state spaces. Fortunately, improved actions can instead be found by applying the formula at line 11 on demand, only for the states where such actions are actually necessary (another alternative, where an explicitly parameterized policy is stored and updated, will be presented in Section III-B). Another observation is that learning is often split in distinct *trials*, instead of being performed along a single trajectory as shown – for simplicity – in Algorithm 1. Each trial can be terminated after a predefined length of time, or upon reaching certain special states (e.g., corresponding to a final goal or to an irrecoverable failure).

### III. ONLINE LSPI WITH PRIOR KNOWLEDGE

RL is usually envisioned as working without any prior knowledge about the system or the solution. However, in practice, prior knowledge is often available, and using it can be very beneficial. Prior knowledge may refer, e.g., to the policy, to the Q-function, or to the system dynamics. We describe a way to exploit prior knowledge about the optimal policy, or more generally about good policies that are not necessarily optimal. Policy knowledge is often easier to obtain than knowledge about the value function (which is a rather complicated function of the system dynamics, reward function, and the policy itself).

Policy knowledge can generally be described by defining constraints. The main benefit of constraining policies is a speedup of the learning process, expected because the algorithm restricts its focus to the constrained class of policies, and no longer invests learning time in trying other, unsuitable policies. We do not focus on accelerating *computation*, but rather on using *experience* more efficiently: an algorithm is fast if it performs well after a observing a small number of transitions. This measure of learning speed is crucial in practice, because obtaining data is costly (in terms of energy consumption, wear-and-tear, and possibly economic profit), whereas computation is relatively cheap.

We develop an online LSPI variant for *globally monotonic policies*. Such policies are monotonic with respect to any state variable, if the other state variables are held constant. Monotonic policies are suitable for controlling important classes

of systems. For instance, policies that are linear in the state variables, and therefore monotonic, work well for controlling (nearly) linear systems, as well as nonlinear systems in neighborhoods of equilibria where they are nearly linear. Monotonic policies are also work suitable for controlling some linear systems with monotonic input nonlinearities (such as saturation or dead-zone nonlinearities), for which the policy may be strongly nonlinear, but still monotonic.

### A. Globally monotonic policies

Consider a system with a  $D$ -dimensional, continuous state space  $X \subset \mathbb{R}^D$ . We assume that  $X$  is a hyperbox:

$$X = [x_{\min,1}, x_{\max,1}] \times \cdots \times [x_{\min,D}, x_{\max,D}] \quad (6)$$

where  $x_{\min,d} \in \mathbb{R}$ ,  $x_{\max,d} \in \mathbb{R}$ , and  $x_{\min,d} < x_{\max,d}$ , for  $d = 1, \dots, D$ . For simplicity, we also assume that  $u$  is scalar, but the entire derivation in the sequel can easily be extended to multiple action variables.

A policy  $h$  is monotonic along the  $d$ th dimension of the state space if and only if, for any pair  $(x, \bar{x}) \in X \times X$  of states that fulfill:

$$x_d \leq \bar{x}_d; \text{ and } x_{d'} = \bar{x}_{d'} \quad \forall d' \neq d$$

the policy satisfies:

$$\delta_{\text{mon},d} \cdot h(x) \leq \delta_{\text{mon},d} \cdot h(\bar{x}) \quad (7)$$

where  $\delta_{\text{mon},d} \in \{-1, 1\}$  specifies the monotonicity direction: if  $\delta_{\text{mon},d}$  is  $-1$  then  $h$  is decreasing along the  $d$ th dimension, and if it is  $1$  then  $h$  is increasing. A policy is (globally) monotonic if it is monotonic along every dimension  $d$ . The monotonicity directions are collected in a vector  $\delta_{\text{mon}} = [\delta_{\text{mon},1}, \dots, \delta_{\text{mon},D}]^T \in \{-1, 1\}^D$ , which encodes the prior knowledge about the policy monotonicity.

### B. Enforcing monotonicity

To efficiently enforce policy monotonicity, two choices are made. The first choice is to represent the policy explicitly, rather than implicitly via the Q-function, as in the original online LSPI. This frees us from translating the monotonicity constraints into Q-function constraints – a task that, while possible in principle, is very difficult to perform in practice, due to the complex relationship between a policy and its Q-function. The monotonicity constraints are enforced directly on the policy parameters, in the policy improvement step. Note that, since continuous-state policies cannot be represented exactly in general, the explicit representation comes at the expense of introducing policy approximation errors.

The second choice is to employ a linear policy parameterization:<sup>1</sup>

$$\hat{h}(x) = \sum_{i=1}^{\mathcal{N}} \varphi_i(x) \vartheta_i = \varphi^T(x) \vartheta \quad (8)$$

where  $\varphi(x) = [\varphi_1(x), \dots, \varphi_{\mathcal{N}}(x)]^T$  are axis-aligned, normalized radial basis functions (RBFs) with their centers arranged

<sup>1</sup>We use calligraphic notation to differentiate mathematical objects related to policy approximation from those related to Q-function approximation (e.g., the policy parameters are denoted by  $\vartheta$ , whereas the Q-function parameters are denoted by  $\theta$ ).

on a grid and having identical widths; and  $\vartheta \in \mathbb{R}^{\mathcal{N}}$  is the policy parameter vector. The first and last grid points are placed at the boundaries of the hyperbox state space (6), and the grid spacing is equidistant along each dimension. The formula to compute the  $i$ th normalized RBF is:

$$\varphi_i(x) = \frac{\bar{\varphi}_i(x)}{\sum_{i'=1}^{\mathcal{N}} \bar{\varphi}_{i'}(x)}, \text{ where } \bar{\varphi}_i(x) = \exp \left[ - \sum_{d=1}^D \frac{(x_d - c_{i,d})^2}{b_{i,d}^2} \right]$$

and where  $c_{i,d}$  is the center coordinate along the  $d$ th dimension, and  $b_{i,d}$  is the width along this dimension.

With this specific policy approximator, in order to satisfy (7) it suffices to enforce a proper ordering of the parameters corresponding to each sequence of RBFs, along all the grid lines and in every dimension of the state space. We have verified the sufficiency of this condition using extensive experimentation, for many RBF configurations and parameter values, and we conjecture that it is also sufficient in general — although we have not yet formally proven this.

To develop a mathematical notation for this condition, denote the grid sizes along each dimension by, respectively,  $\mathcal{N}_1, \dots, \mathcal{N}_D$ ; there are  $\mathcal{N} = \prod_{d=1}^D \mathcal{N}_d$  RBFs in total. Furthermore, denote by  $\varphi_{i_1, \dots, i_D}$  the RBF located at grid indices  $i_1, \dots, i_D$ , and by  $\vartheta_{i_1, \dots, i_D}$  the parameter that multiplies this RBF in (8). We will use these  $D$ -dimensional indices interchangeably with the single-dimensional indices that appear in (8). Choosing any bijective mapping between the  $D$ -dimensional indices and the single-dimensional ones suffices to make these two indexing conventions equivalent.

The monotonicity conditions on the parameters can now be written in the following, linear form:

$$\begin{aligned} \delta_{\text{mon},1} \cdot \vartheta_{1,i_2,i_3,\dots,i_D} &\leq \delta_{\text{mon},1} \cdot \vartheta_{2,i_2,i_3,\dots,i_D} \leq \dots \\ &\dots \leq \delta_{\text{mon},1} \cdot \vartheta_{\mathcal{N}_1,i_2,\dots,i_D} \text{ for all } i_2, i_3, \dots, i_D, \\ \delta_{\text{mon},2} \cdot \vartheta_{i_1,1,i_3,\dots,i_D} &\leq \delta_{\text{mon},2} \cdot \vartheta_{i_1,2,i_3,\dots,i_D} \leq \dots \\ &\dots \leq \delta_{\text{mon},2} \cdot \vartheta_{i_1,\mathcal{N}_2,i_3,\dots,i_D} \text{ for all } i_1, i_3, \dots, i_D, \\ &\dots \quad \dots \quad \dots \\ \delta_{\text{mon},D} \cdot \vartheta_{i_1,i_2,i_3,\dots,1} &\leq \delta_{\text{mon},D} \cdot \vartheta_{i_1,i_2,i_3,\dots,2} \leq \dots \\ &\dots \leq \delta_{\text{mon},D} \cdot \vartheta_{i_1,i_2,i_3,\dots,\mathcal{N}_D} \text{ for all } i_1, i_2, \dots, i_{D-1} \end{aligned} \quad (9)$$

The total number of inequalities in (9) (height of  $\Delta_{\text{mon}}$ ) is:

$$\sum_{d=1}^D \left( (\mathcal{N}_d - 1) \prod_{d'=1, d' \neq d}^D \mathcal{N}_{d'} \right)$$

This type of constraints is easier to understand in the two-dimensional case. For instance, an ordering corresponding to a  $3 \times 3$  grid of RBFs could be:

$$\begin{aligned} \vartheta_{1,1} &\leq \vartheta_{1,2} \leq \vartheta_{1,3} \\ &\geq \quad \geq \quad \geq \\ \vartheta_{2,1} &\leq \vartheta_{2,2} \leq \vartheta_{2,3} \\ &\geq \quad \geq \quad \geq \\ \vartheta_{3,1} &\leq \vartheta_{3,2} \leq \vartheta_{3,3} \end{aligned} \quad (10)$$

in which case the policy would be decreasing along the first dimension of  $X$  – vertically in (10) – and increasing along the second dimension – horizontally in (10).

The constraints (9) can be collected in a concise matrix form:

$$\Delta_{\text{mon}} \vartheta \leq \mathbf{0} \quad (11)$$

where each row represents a constraint, and  $\mathbf{0}$  is an appropriately sized vector of zeros. For example, the constraints (10) lead to the following matrix  $\Delta_{\text{mon}}$ :

$$\begin{bmatrix} \vartheta_{1,1} \leq \vartheta_{1,2} & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ \vartheta_{1,2} \leq \vartheta_{1,3} & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ \vartheta_{2,1} \leq \vartheta_{2,2} & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ \vartheta_{2,2} \leq \vartheta_{2,3} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ \vartheta_{3,1} \leq \vartheta_{3,2} & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ \vartheta_{3,2} \leq \vartheta_{3,3} & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ \vartheta_{1,1} \geq \vartheta_{2,1} & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vartheta_{2,1} \geq \vartheta_{3,1} & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vartheta_{1,2} \geq \vartheta_{2,2} & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ \vartheta_{2,2} \geq \vartheta_{3,2} & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ \vartheta_{1,3} \geq \vartheta_{2,3} & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ \vartheta_{2,3} \geq \vartheta_{3,3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

The constraints along the horizontal axis of (10) are added to  $\Delta_{\text{mon}}$  first, followed by the constraints along the vertical axis. The column order of  $\Delta_{\text{mon}}$  depends on the choice of the bijection between two-dimensional and single-dimensional parameter indices (see discussion above). In this particular case, the parameters have been collected column-wise, so that their single-dimensional order is: (1, 1), (2, 1), (3, 1), (1, 2), (2, 2), (3, 2), (1, 3), (2, 3), (3, 3). To help with reading the matrix, the corresponding inequality constraint is added to the left of each row.

An added benefit of the approximate policy (8) is that it produces continuous actions. Note, however, that if a discrete-action Q-function approximator is employed – as is often the case, and as we also do in Section IV below – the continuous actions given by the policy must be *discretized* during learning into actions belonging to the discrete action set, denoted by  $U_{\text{d}} = \{u_1, \dots, u_M\}$ . (This is because a discrete-action approximator will return an identically zero basis function vector when presented with an action not on the discretization.) For instance, one could use nearest-neighbor discretization:

$$u_{\text{d}} = \arg \min_{j=1, \dots, M} |u - u_j| \quad (12)$$

where  $u_{\text{d}}$  denotes the discretized action (recall the action was assumed to be scalar; for multiple action variables, a vector norm should be used instead of the absolute value). In this case, the policy evaluation step actually estimates the Q-function of a discretized version of the policy.

### C. Online LSPI with monotonic policies

The prior knowledge about policy monotonicity is employed in online LSPI by replacing the unconstrained policy improvement (line 11 of Algorithm 1) with the constrained least-

squares problem:

$$\vartheta_{\tau+1} = \arg \min_{\vartheta \text{ satisfying (9)}} \sum_{i_s=1}^{\mathcal{N}_s} (\varphi^{\text{T}}(x_{i_s}) \vartheta - u_{i_s})^2$$

where  $u_{i_s} = \arg \max_u \phi^{\text{T}}(x_{i_s}, u) \theta_{\tau}$  (13)

Here,  $\{x_1, \dots, x_{\mathcal{N}_s}\}$  is an arbitrary set of state samples to be used for policy improvement. Since the constraints (9) are linear, the problem (13) can be efficiently solved using quadratic programming. The parameter vector  $\vartheta_{\tau+1}$  leads to a monotonic and approximately improved policy  $\hat{h}_{\tau+1}(x) = \varphi^{\text{T}}(x) \vartheta_{\tau+1}$ , which is used instead of the unconstrained policy to choose actions and in the updates of  $\Gamma$ .

Algorithm 2 summarizes online LSPI with monotonic policies, a general linear parameterization of the Q-function, and  $\varepsilon$ -greedy exploration.

---

#### Algorithm 2 Online LSPI with monotonic policies.

---

**Input:** Q-function BFs  $\phi_1, \dots, \phi_n$ , policy BFs  $\varphi_1, \dots, \varphi_{\mathcal{N}}$ ; monotonicity directions  $\delta_{\text{mon}}$ ; samples  $\{x_1, \dots, x_{\mathcal{N}_s}\}$ ;  $\gamma$ ;  $K_{\theta}$ ;  $\{\varepsilon_k\}_{k=0}^{\infty}$ ;  $\delta$

- 1:  $\tau \leftarrow 0$ ; initialize policy parameter  $\vartheta_0$
- 2:  $\Gamma \leftarrow \delta I_{n \times n}$ ;  $z \leftarrow 0$
- 3: measure initial state  $x_0$
- 4: **for** each time step  $k \geq 0$  **do**
- 5:  $u_k \leftarrow \begin{cases} \varphi^{\text{T}}(x_k) \vartheta_{\tau} & \text{w.p. } 1 - \varepsilon_k \\ \text{a uniform random action in } U & \text{w.p. } \varepsilon_k \end{cases}$
- 6: apply  $u_k$ , measure next state  $x_{k+1}$  and reward  $r_{k+1}$
- 7:  $\Gamma \leftarrow \Gamma + \phi(x_k, u_k) \phi^{\text{T}}(x_k, u_k) - \gamma \phi(x_k, u_k) \phi^{\text{T}}(x_{k+1}, \varphi^{\text{T}}(x_{k+1}) \vartheta_{\tau})$
- 8:  $z \leftarrow z + \phi(x_k, u_k) r_{k+1}$
- 9: **if**  $k = (\tau + 1)K_{\theta}$  **then**
- 10: find  $\theta_{\tau}$  by solving  $\frac{1}{k+1} \Gamma \theta_{\tau} = \frac{1}{k+1} z$
- 11: find  $\vartheta_{\tau+1}$  by solving (13)
- 12:  $\tau \leftarrow \tau + 1$
- 13: **end if**
- 14: **end for**

---

To generalize this framework to multiple action variables, a distinct policy parameter vector should be used for every action variable, and the monotonicity constraints should be enforced separately, for each of these parameter vectors. Different monotonicity directions can be imposed for different action variables.

As an alternative to (13), policy improvement could be performed with:

$$\begin{aligned} \vartheta_{\tau+1} &= \arg \max_{\vartheta \text{ satisfying (9)}} \sum_{i_s=1}^{\mathcal{N}_s} \widehat{Q}_{\tau}(x_{i_s}, \varphi^{\text{T}}(x_{i_s}) \vartheta) \\ &= \arg \max_{\vartheta \text{ satisfying (9)}} \sum_{i_s=1}^{\mathcal{N}_s} \phi^{\text{T}}(x_{i_s}, \varphi^{\text{T}}(x_{i_s}) \vartheta) \theta_{\tau} \end{aligned}$$

which aims to maximize the approximate Q-values of the actions chosen by the policy in the state samples. This is a more direct way of improving the policy, but unfortunately

it is generally a very difficult nonlinear optimization problem. Since the samples  $u_{i_s}$  are already fixed when  $\vartheta_{\tau+1}$  is computed, the optimization problem (13) is convex, and thus easier to solve.

#### IV. EXPERIMENTAL STUDY

In this section, we investigate the effects of using prior knowledge in online LSPI. To this end, in a simulation example involving the stabilization of a DC motor, we compare the learning performance of online LSPI with prior knowledge (Algorithm 2), with the performance of the original online LSPI (Algorithm 1), which does not use prior knowledge.

##### A. DC motor problem and some near-optimal solutions

The DC motor is described by the discrete-time dynamics:

$$f(x, u) = Ax + Bu$$

$$A = \begin{bmatrix} 1 & 0.0049 \\ 0 & 0.9540 \end{bmatrix}, \quad B = \begin{bmatrix} 0.0021 \\ 0.8505 \end{bmatrix}$$

where  $x_1 = \alpha \in [-\pi, \pi]$  rad is the shaft angle,  $x_2 = \dot{\alpha} \in [-16\pi, 16\pi]$  rad/s is the angular velocity, and  $u \in [-10, 10]$  V is the control input (voltage). The state variables are restricted to their domains using saturation. The goal is to stabilize the system around  $x=0$ , and is described by the quadratic reward function:

$$\rho(x, u) = -x^T Q_{\text{rew}} x - R_{\text{rew}} u^2$$

$$Q_{\text{rew}} = \begin{bmatrix} 5 & 0 \\ 0 & 0.01 \end{bmatrix}, \quad R_{\text{rew}} = 0.01$$

with discount factor  $\gamma = 0.98$ .

Because the dynamics are linear and the reward function is quadratic, the optimal policy would be a linear state feedback, of the form  $h(x) = L^T x$ , if the constraints on the state and action variables were disregarded. The gain vector  $L$  can be computed from  $f$  and  $\rho$ , using an extension of linear quadratic control to the discounted case, as explained, e.g., in, Section 3.2 of [3]. The result is  $L = [-12.93, -0.69]^T$ , corresponding to a policy that monotonically decreases along both axes of the state space. This monotonicity property will be used in the sequel. Note that only prior knowledge about the *signs* of the feedback gains is required to establish the policy monotonicity directions, and the actual *values* of these gains are not needed.

Figure 1(a) presents the linear policy corresponding to  $L$ , after additionally restricting the control action to the allowed interval  $[-10, 10]$ , using saturation. For comparison, Figure 1(b) presents a near-optimal policy, computed by an interpolation-based approximate value iteration algorithm, with an accurate approximator. This algorithm takes into account that the states and the action are constrained to finite domains. Note that value iteration employs a model, whereas online LSPI does not. The policy from Figure 1(b) strongly resembles Figure 1(a), and is monotonic over most of the state space. The only nonmonotonic regions appear in the top-left and bottom-right corners of the figure, probably because the state constraints. We conclude that the class of monotonic policies to which online LSPI will be restricted does indeed contain near-optimal solutions.

##### B. Policy and Q-function approximators, parameter settings, and performance criterion

To apply online LSPI with monotonicity constraints, the policy is represented using a grid of RBFs, as described in Section III-B. The grid contains  $9 \times 9$  RBFs, so the policy has 81 parameters. The RBF width along each dimension is identical to the distance between two adjacent RBFs along that dimension. To perform the policy improvements (13),  $\mathcal{N}_s = 1000$  uniformly distributed, random state samples are used.

The Q-function approximator relies on the same grid of state-dependent RBFs as the policy approximator, and on a discretization of the action space into 3 discrete values:  $\{-10, 0, 10\}$ . (Of course, in general the Q-functions BFs can be chosen independently from the policy BFs.) To obtain the state-action BFs required for Q-function approximation (4), the RBFs are replicated for every discrete action, obtaining a total of  $81 \cdot 3 = 243$  BFs. When computing approximate Q-values, all the BFs that do not correspond to the current discrete action are taken equal to 0, i.e., the vector of Q-function BFs is  $\phi(x, u) = [\mathcal{I}(u = -10) \cdot \varphi^T(x), \mathcal{I}(u = 0) \cdot \varphi^T(x), \mathcal{I}(u = 10) \cdot \varphi^T(x)]^T$ , where the indicator function  $\mathcal{I}$  is 1 when its argument is true, and 0 otherwise.

Note that, although the parameterized policy produces continuous actions, the Q-function approximator only works for the discrete actions considered. Therefore, the continuous actions produced by the policy must be discretized during learning, at lines 5 and 7 of Algorithm 2. We employ nearest-neighbor discretization (12) for this purpose.

The learning experiment has a length of 600 s and is divided into learning trials having the same length, 1.5 s. The initial state of each trial is chosen randomly from a uniform distribution over the state space. The policy is improved once every  $K_\theta = 100$  transitions. An exponentially decaying exploration schedule is used that starts from an initial probability  $\epsilon_0 = 1$ , and decays so that after  $t = 200$  s,  $\epsilon$  becomes 0.1. The parameter  $\delta$  is set to 0.001. The initial policy parameters, together with the resulting policy, are identically zero.

The original online LSPI employs the same Q-function approximator and settings as online LSPI with prior knowledge, but does not approximate the policy or enforce monotonicity constraints. Instead, it computes greedy actions on demand, by maximizing the Q-function (see Section II). The initial policy chooses the first discrete action ( $-10$ ) for any state.

After each online LSPI experiment is completed, snapshots of the policy taken at increasing moments of time are evaluated. This produces a curve recording the control performance of the policy over time. During performance evaluation, learning and exploration are turned off. Policies are evaluated using simulation, by estimating their average return (score) over the grid of initial states  $X_0 = \{-\pi, -\pi/2, 0, \pi/2, \pi\} \times \{-10\pi, -5\pi, -2\pi, -\pi, 0, \pi, 2\pi, 5\pi, 10\pi\}$ . The return from each state on this grid is estimated by simulating only the first  $K$  steps of the controlled trajectory, with  $K$  chosen large enough to guarantee the estimate is within a 0.1 distance of the true return.

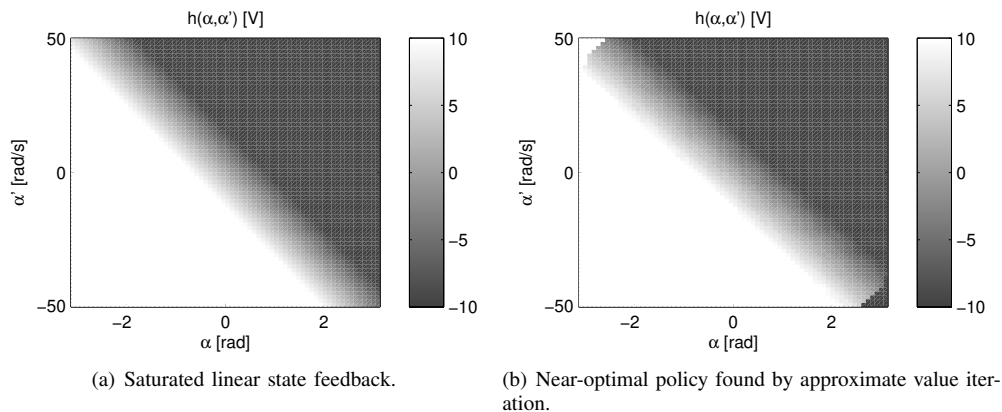


Fig. 1. Near-optimal policies for the DC motor.

### C. Results and discussion

Figure 2 shows the learning performance of online LSPI with monotonic policies, in comparison to the performance of the original online LSPI. Mean values across 20 independent runs are reported, together with 95% confidence intervals on these means.

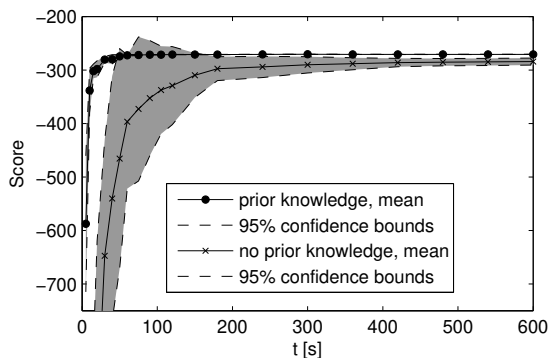


Fig. 2. Performance comparison between online LSPI with prior knowledge and the original online LSPI. The horizontal axis shows the time spent interacting with the system (simulation time).

Using prior knowledge leads to much faster and more reliable learning: the score converges in around 50 s of simulation time, during which 10000 samples are observed. In contrast, online LSPI without prior knowledge requires more than 300 s (60000 samples) to reach a near-optimal performance, and has a larger variation in performance across the 20 runs, which can be seen in the wider 95% confidence intervals.

The final performance is also better when using monotonic policies, largely because these policies output continuous actions. Recall however from Section IV-B that this advantage cannot be exploited *during* learning, when the actions must be discretized to make them compatible with the Q-function approximator. To better understand the effects of discretizing actions, Figure 3 shows the performance of the policies computed using online LSPI with prior knowledge *and discretized*, in comparison to the original online LSPI. While the final performance is now the same, the learning speed advantage of

using prior knowledge is evidently maintained.

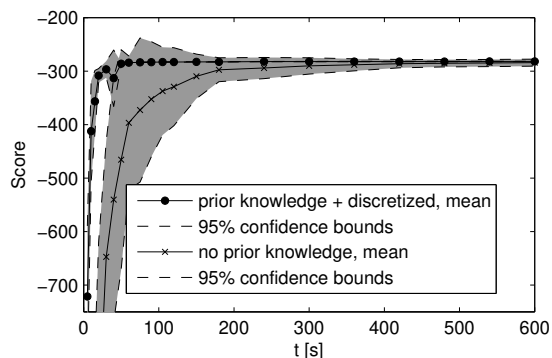
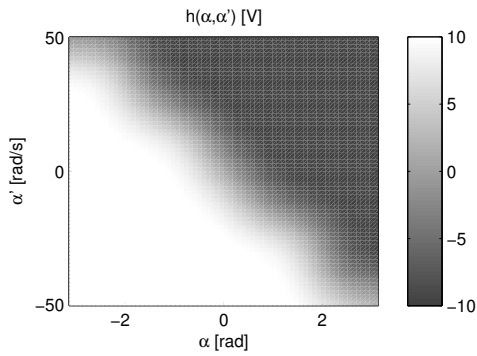


Fig. 3. Performance comparison between online LSPI with prior knowledge and discretized policies, and the original online LSPI.

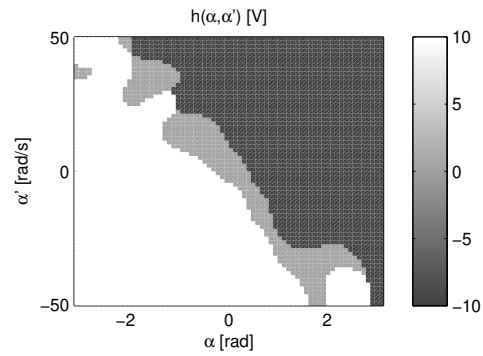
Figure 4 compares a representative solution obtained using prior knowledge with one obtained by the original online LSPI. The policy of Figure 4(b), obtained without using prior knowledge, violates monotonicity in several areas. The Q-function in Figure 4(c), corresponding to the monotonic policy, is smoother. The control performance of the monotonic policy – Figure 4(e) – is better than the performance of the policy found without prior knowledge – Figure 4(f). This difference appears mainly because the monotonic policy outputs continuous actions.

The mean execution time of online LSPI with prior knowledge is 1046.5 s, with a 95% confidence interval of [1024.2, 1068.9] s. For the original online LSPI algorithm, the mean execution time is 87.7 s with a confidence interval of [81.8, 93.6] s. These execution times were recorded while running the algorithms in MATLAB 7 on a PC with an Intel Core 2 Duo E6550 2.33 GHz CPU and with 3 GB of RAM.

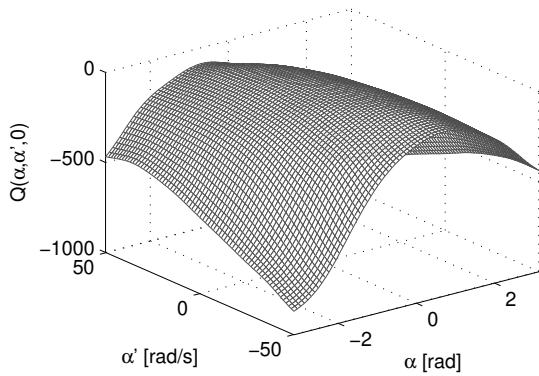
So, although online LSPI with prior knowledge learns faster in terms of *simulation* time (number of transition samples observed), its *execution* time is larger. This is mainly because the constrained policy improvements (13) are more computationally demanding than the original policy improvements (3). In particular, solving (13) takes much longer than a



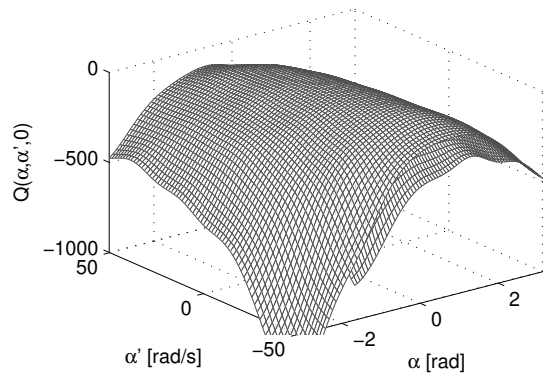
(a) Policy found using prior knowledge.



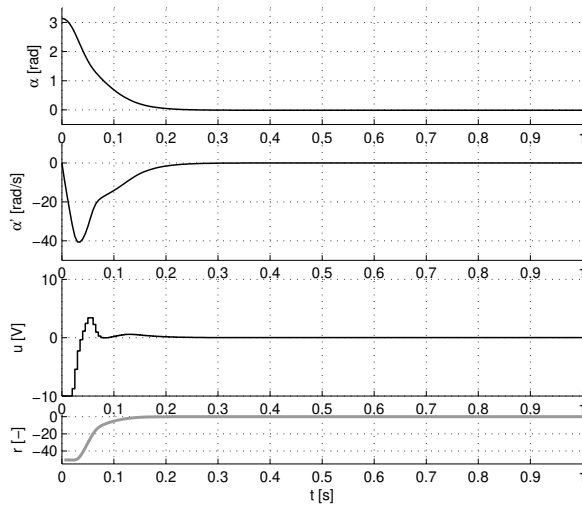
(b) Policy found without prior knowledge.



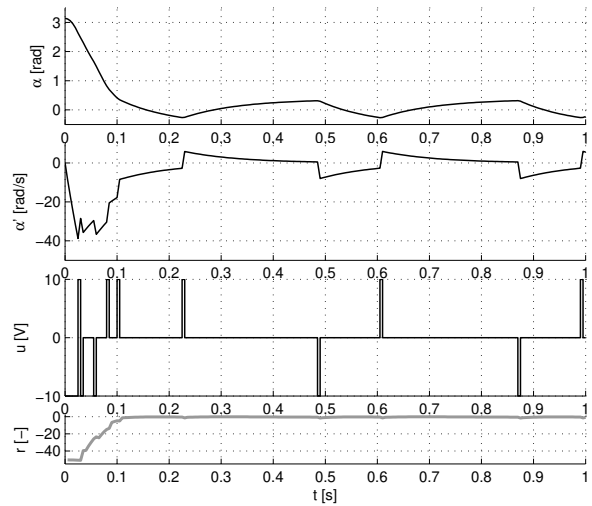
(c) Q-function found using prior knowledge. A slice through the Q-function is shown, taken for  $u = 0$ .



(d) Q-function found without prior knowledge.



(e) Trajectory controlled by the policy found using prior knowledge.



(f) Trajectory controlled by the policy found without prior knowledge.

Fig. 4. Left: Representative solutions found using prior knowledge (with continuous-action, parameterized policies). Right: Representative solutions found without prior knowledge (with discrete-action, implicitly represented policies).

sampling period (around 0.75 s, whereas  $T_s = 0.005$  s), which means that the algorithm cannot be directly applied in real-time. To address this difficulty, besides the obvious solution of optimizing the implementation (e.g., by switching from MATLAB to C, which should provide a significant boost in execution speed), another possibility is to perform the policy improvements *asynchronously*, on a different thread than the one responsible with controlling the system. This thread could run on another processor. While executing policy improvement, the system should be controlled with the previously available policy, possibly collecting transition samples for later use in evaluating the new policy.

## V. CONCLUSIONS AND OPEN ISSUES

We have described and empirically studied an approach to integrate prior knowledge into online least-squares policy iteration. We have focused on a particular type of prior knowledge: that a (near-)optimal policy is monotonic in the state variables. Such policies are appropriate in certain problems of practical interest, such as (nearly) linear systems, or nonlinear systems arising from linear dynamics combined with monotonic input nonlinearities. For an example involving the stabilization of a DC motor, using this type of prior knowledge has led to more reliable and faster learning (in terms of time spent interacting with the system).

The global monotonicity requirement is restrictive in general, so an important point of improvement is the combination of online LSPI with more general types of prior knowledge. Some immediate improvements are enforcing monotonicity only with respect to a subset of state variables, or only over a subregion of the state space, such as in the neighborhood of an equilibrium. A very general way to express prior knowledge are inequality and equality constraints of the form  $g_{in}(x, h(x)) \leq 0$ ,  $g_{eq}(x, h(x)) = 0$ . Unlike the monotonicity property, such constraints can be exploited without representing the policy explicitly. Instead, they can be enforced while computing improved actions on demand with (3).

It would also be useful to empirically investigate the effects of using continuous-action Q-function approximators in online LSPI with monotonic policies. This can improve performance by eliminating the need to discretize the actions during learning. An efficient way of using continuous actions is described by [19].

Finally, while here we used the LSPI algorithm as a basis for our extensions, several other recently developed policy iteration algorithms based on least-squares methods (see [13] for a survey) could also benefit from prior knowledge.

## REFERENCES

- [1] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.
- [2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.
- [3] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Athena Scientific, 2007, vol. 2.
- [4] L. Buşoniu, R. Babuška, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*, ser. Automation and Control Engineering. Taylor & Francis CRC Press, 2010.
- [5] J. Boyan, "Technical update: Least-squares temporal difference learning," *Machine Learning*, vol. 49, pp. 233–246, 2002.
- [6] A. Nedić and D. P. Bertsekas, "Least-squares policy evaluation algorithms with linear function approximation," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 13, no. 1–2, pp. 79–110, 2003.
- [7] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.
- [8] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, pp. 503–556, 2005.
- [9] H. Yu and D. P. Bertsekas, "Convergence results for some temporal difference methods based on least squares," *IEEE Transactions on Automatic Control*, vol. 54, no. 7, pp. 1515–1531, 2009.
- [10] C. Thiery and B. Scherrer, "Least-squares  $\lambda$  policy iteration: Bias-variance trade-off in control problems," in *Proceedings 27th International Conference on Machine Learning (ICML-10)*, Haifa, Israel, 21–24 June 2010, pp. 1071–1078.
- [11] D. P. Bertsekas, "Approximate dynamic programming," 20 November 2010, update of Chapter 6 in volume 2 of the book *Dynamic Programming and Optimal Control*. Available at <http://web.mit.edu/dimitrib/www/dpchapter.html>.
- [12] L. Buşoniu, A. Lazaric, M. Ghavamzadeh, R. Munos, R. Babuška, and B. De Schutter, "Least-squares methods for policy iteration," in *Reinforcement Learning: State of the Art*, M. Wiering and M. van Otterlo, Eds. Springer, 2011, submitted.
- [13] D. P. Bertsekas, "Approximate policy iteration: A survey and some new methods," *Journal of Control Theory and Applications*, vol. 9, no. 3, pp. 310–335, 2011.
- [14] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, "Online least-squares policy iteration for reinforcement learning control," in *Proceedings 2010 American Control Conference (ACC-10)*, Baltimore, US, 30 June – 2 July 2010, pp. 486–491.
- [15] R. S. Sutton, "Learning to predict by the method of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.
- [16] L. Li, M. L. Littman, and C. R. Mansley, "Online exploration in least-squares policy iteration," in *Proceedings 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-09)*, vol. 2, Budapest, Hungary, 10–15 May 2009, pp. 733–739.
- [17] T. Jung and D. Polani, "Kernelizing LSPE( $\lambda$ )," in *Proceedings 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL-07)*, Honolulu, US, 1–5 April 2007, pp. 338–345.
- [18] L. Buşoniu, B. De Schutter, R. Babuška, and D. Ernst, "Using prior knowledge to accelerate online least-squares policy iteration," in *2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR-10)*, Cluj-Napoca, Romania, 28–30 May 2010.
- [19] J. Pavis and M. Lagoudakis, "Binary action search for learning continuous-action control policies," in *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, Montreal, Canada, 14–18 June 2009, pp. 793–800.