Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Least-squares policy iteration algorithms for robotics: Online, continuous, and automatic[☆]

Stefan R. Friedrich[a,b,*], Michael Schreibauer[a], Martin Buss[a,b]

[a] Technical University of Munich, Department of Electrical and Computer Engineering, Chair of Automatic Control Engineering (LSR), Theresienstr. 90, 80333 Munich, Germany
[b] Technical University of Munich, Institute for Advanced Study, Lichtenbergstr. 2a, 85748 Garching, Germany

ARTICLE INFO

Keywords:
Reinforcement learning
Policy iteration
Continuous actions
Robotics
Sparsification

ABSTRACT

Reinforcement learning (RL) is a general framework to acquire intelligent behavior by trial-and-error and many successful applications and impressive results have been reported in the field of robotics. In robot control problem settings, it is oftentimes characteristic that the algorithms have to learn online through interaction with the system while it is operating, and that both state and action spaces are continuous. Least-squares policy iteration (LSPI) based approaches are therefore particularly hard to employ in practice, and parameter tuning is a tedious and costly enterprise. In order to mitigate this problem, we derive an automatic online LSPI algorithm that operates over continuous action spaces and does not require an a-priori, hand-tuned value function approximation architecture. To this end, we first show how the kernel least-squares policy iteration algorithm can be modified to handle data online by recursive dictionary and learning update rules. Next, borrowing sparsification methods from kernel adaptive filtering, the continuous action-space approximation in the online least-squares policy iteration algorithm can be efficiently automated as well. We then propose a similarity-based information extrapolation for the recursive temporal difference update in order to perform the dictionary expansion step efficiently in both algorithms. The performance of the proposed algorithms is compared with respect to their batch or hand-tuned counterparts in a simulation study. The novel algorithms require less prior tuning and data is processed completely on the fly, yet the results indicate that similar performance can be obtained as by careful hand-tuning. Therefore, engineers from both robotics and AI can benefit from the proposed algorithms when an LSPI algorithm is faced with online data collection and tuning by experiment is costly.

1. Introduction

For many robotic tasks detailed mathematical modeling is hard or time-consuming, which makes reinforcement learning (RL) an attractive alternative to model-based control design. Interacting with the environment in trial-and-error fashion is the core idea of RL methods (Sutton and Barto, 1998), allowing to infer desired behavior. While RL constitutes a general framework to learn sophisticated behaviors in a multitude of disciplines, robotic tasks are often closely related to optimal or adaptive control problems. In this context, some RL methods can be conceived of as direct adaptive optimal control (Sutton et al., 1992). Some contributions in the field of adaptive dynamic programming are also relevant, particularly if it is important to keep a continuous-time formulation, see for example (Vrabie et al., 2012) and the references therein. For robot control, iterative discrete-time RL algorithms are more frequently used, see Kober et al. (2013) for a comprehensive overview. Selected examples include tracking performance improvement of robot manipulators (Pane et al., 2019; Friedrich and Buss, 2017), aerial transportation with drones (Palunko et al., 2013), control of autonomous vehicles (Wang et al., 2014; Vankadari et al., 2018), marine vehicle navigation (Tziortziotis et al., 2016), pendulum systems (Xu et al., 2014; Deisenroth et al., 2015), human–robot cooperative manipulation (Palunko et al., 2014), apprenticeship learning (Mori et al., 2011) and information exchange in cooperative multi-agent systems (Tolić and Palunko, 2017).

The two main classes of RL algorithms are value-based approaches and value function free methods, e. g., policy search. On the one hand, policy based RL is predominant in robotic applications due to several factors (Deisenroth et al., 2013): a policy search algorithm

☆ No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.engappai.2019.04.001.

* Corresponding author at: Technical University of Munich, Department of Electrical and Computer Engineering, Chair of Automatic Control Engineering (LSR), Theresienstr. 90, 80333 Munich, Germany.

E-mail addresses: s.friedrich@tum.de (S.R. Friedrich), m.schreibauer@tum.de (M. Schreibauer), mb@tum.de (M. Buss).

https://doi.org/10.1016/j.engappai.2019.04.001
Received 15 June 2018; Received in revised form 21 February 2019; Accepted 1 April 2019
Available online 31 May 2019

works with an explicitly pre-structured parametric policy and iteratively improves the policy by locally optimizing directly in the space of parameters. Therefore, suitable policy representations allow to reduce the learning problem from the potentially high-dimensional state–action space to a lower-dimensional optimization problem in parameter space, greatly simplifying the learning problem in practice (Stulp and Sigaud, 2013). Moreover, the demand for continuous and possibly multidimensional action spaces is more naturally covered in policy based algorithms. On the other hand, a value function based method constructs a ranking over the state and action sets w. r. t. the expected long-term reward, thereby implicitly encoding a globally optimal policy. This approach, however, entails properties that become particularly problematic for robot control (Deisenroth et al., 2013). Function approximators (Geramifard et al., 2013) must be employed to represent the value of a given state/action combination in the oftentimes large state–action space of robotic systems. Accordingly, the computational complexity easily becomes intractable due to the curse of dimensionality. A particularly recurring research question is therefore how the action space in continuous domains can be smoothly approximated, e. g., by discretization and subsequent symbolic post-processing (Alibekov et al., 2018) or heuristically by expert knowledge and fuzzy representations (Hourfar et al., 2019).

Despite their drawbacks, value function based algorithms are preferred in some robotic applications in order to avoid the limitations of policy search, see Kober et al. (2013, Tab. 1). In particular, one needs to construct suitable policy parameterizations and find good initial policy parameters for local optimization in policy search. A class of popular value function algorithms is based on least-squares policy iteration (LSPI) (Lagoudakis and Parr, 2003). Extensions to approximation-based LSPI are studied in detail in Bušoniu et al. (2010), and an online least-squares policy iteration (OLSPI) algorithm is derived in Buşoniu et al. (2010). These algorithms iteratively evaluate and improve the control policy, are sample-efficient, and have comparatively good convergence properties due to the least-squares techniques for policy evaluation. For example, Palunko et al. (2013), Vankadari et al. (2018), Palunko et al. (2014), Tolić and Palunko (2017), Wang et al. (2014), Tziortziotis et al. (2016) all employ some form of LSPI.

It is currently, however, rather tedious to apply LSPI algorithms to practical robotic problems. First of all, there often is a demand not only for a continuous state but also a continuous action space representation. Therefore, it is necessary to employ a value function approximation (VFA) method and the achievable performance depends considerably on an appropriate representation for the system at hand. Next, operating online means that data cannot be collected in advance but has to be obtained incrementally, requiring fast enough processing cycle times and manageable memory complexity. Finally, it is crucial to employ well-tuned algorithmic parameters in order to obtain a performant learning system. For example, Anderlini et al. (2017) report unexpected behavior of LSPI in the control of a wave energy converter model, presumably due the radial basis function approximation. In robotics, this issue can become even more tedious, particularly when tuning the algorithmic parameters is costly in experimental setups where merely collecting suitable data can be hard, e. g., in closed-loop feedback systems. In summary, to leverage the potential of LSPI in robotics, algorithms are needed that operate *online*, over *continuous state and action spaces*, and *automatically* handle the VFA.

1.1. Related work

Given the wealth of literature on general RL, we mostly restrict attention specifically to LSPI class algorithms employing function approximators to represent the value function. A more extensive treatment of approximation-based RL can be found in Bušoniu et al. (2010). If for example deterministic dynamics can be exploited, fuzzy techniques (Bušoniu et al., 2010, Ch. 4) offer a viable alternative to encode prior expert domain knowledge in the value function. An introduction to RL with linear function approximators in particular is provided in Geramifard et al. (2013). In general, however, feature or basis function (BF) selection and correspondingly "a memory management scheme for LSPI's data [...] is non-trivial" (Geramifard et al., 2013, Ch. 4.5, p. 437). From our perspective, adaptively growing kernel representations (Schölkopf and Smola, 2002) offer a promising way to deal with this problem: the very same issue of BF selection with memory management arises in kernel adaptive filtering (Liu et al., 2011), and a multitude of sparsification schemes have recently been developed in the signal processing community. The general VFA problem is pervasive in high-dimensional RL; hence, we omit an in-depth survey of the extensive literature on VFA in favor of reviewing kernel-based RL methods. For a broader perspective, the interested reader is instead referred to the discussion in Sutton and Barto (1998, Ch. 8), Bušoniu et al. (2010, Ch. 3.6), Geramifard et al. (2013, Ch. 3), and the references therein.

Kernel methods (Schölkopf and Smola, 2002) have in common that a sparsified set of features is used to represent a high-dimensional, implicit feature space only by means of the raw data transformed by the kernel. With the versatility of Gaussian processes (Rasmussen and Williams, 2006), kernel methods are also becoming successful more and more in the field of RL. Several methods exploit such a representation to model the dynamics, e.g., (Deisenroth et al., 2015; Polydoros and Nalpantidis, 2017; Vinogradska et al., 2018). We refrain from reviewing these approaches in more detail as they pursue an indirect, i. e., model-based, approach.

Several value-based model-free RL methods with non-parametric value function modeling have been developed, as reviewed next. The paper Ormoneit and Sen (2002) is an early contribution showing that the distribution of the estimate may be conceived of as a Gaussian process. Jung and Polani (2007) further develop kernel least-squares policy evaluation (KLSPE), a kernelized online policy evaluation scheme and demonstrate their results on a high-dimensional benchmark system; however, a discrete set of pre-defined actions is used. Xu et al. (2007) develop kernel-based least-squares policy iteration (KLSPI), a flavor of LSPI where data is selected according to an approximate linear dependency (ALD) criterion and the value function is represented by means of a kernel expansion. Closely related papers are Jakab and Csató (2015) and Yahyaa and Manderick (2014), which employ direct recursive versions of KLSTD respectively KLSPI. These algorithms, however, are not optimized for online usage and are only applicable to discrete state sets. Recently, Cui et al. (2017) demonstrate that so-called Kernel dynamic policy programming (KDPP) is applicable to high-dimensional robotic systems and the authors also compare to the KLSPI algorithm; nonetheless, Cui et al. (2017) uses ALD for the dictionary sparsification step as well and also KDPP is only applicable with a discrete action set. These approaches have the common advantage that the features are generated in data-driven fashion but the VFA is still in linear form. A comparison of these value-based model-free algorithms is summarized in Table 1. As can be seen, the current kernel-trick based approaches lack the capability of continuous action space representation.

A unifying view of kernel-based RL w. r. t. other regularization schemes is given by Taylor and Parr (2009). Another related algorithm is called kernel-based dual heuristic programming (KDHP) (Xu et al., 2013), whose applicability to hardware was shown in Xu et al. (2014) using inverted pendulum systems. Its *online* mechanism, however, is to run RL over simulated data and then use the final policy on the robotic system, which contradicts our requirements outlined above. Xu et al. (2016) compare a batch KLSPI algorithm for unmanned ground vehicle control with an online actor-critic based on KDHP. Along the same lines is the more recent (Huang et al., 2017), using a kernelized RL algorithm for longitudinal control of autonomous land vehicles, operating with batch samples and ALD sparsification as well. Wang et al. (2014) in turn approach the problem of cruise control of an autonomous vehicle by tuning the parameters of a proportional–integral controller online according to a policy learned with KLSPI. In their approach, the data

Table 1
Overview of model-free value-based RL algorithms with kernel VFA capability, with LSPI and OLSPI included for comparison. The symbols ✓, (✓), and ✗ correspond to "yes", "partially", and "no".

Approach		On-line	Continuous		Feature selection		Admiss. system dimensionality	Kernel trick	Initial policy required
Ref.	Algorithm		State	Action	Automatic	Sparsification			
Lagoudakis and Parr (2003)	LSPI	✗	✓	✗	✗	–	low	✗	✗
Buşoniu et al. (2010)	OLSPI	✓	✓	✗	✗	–	low	✗	✗
Bušoniu et al. (2010)	OLSPI with cont. action	✓	✓	✓	✗	–	low	✗	✗
Xu et al. (2007)	KLSPI	✗	✓	✗	✓	ALD	mid	✓	✓
Yahyaa and Manderick (2014)	Online KBLSPI	(✓)	(✓)	✗	✓	ALD	mid	✓	✗
Jakab and Csató (2015)	KRLSTD	✗	(✓)	✗	✓	Laplacian	low	✓	✓
Cui et al. (2017)	KDPP	✗	✓	✗	✓	ALD	high	✓	✗
here	OKLSPI	✓	✓	✗	✓	Coherence	mid	✓	✗
here	AOLSPI	✓	✓	✓	✓	Coherence	mid	(✓)	✗

samples are also collected in advance and the policy is obtained by running the batch algorithm offline.

Pioneering work to analyze the convergence of KLSPI type algorithms for large-scale or continuous state space markov decision processs (MDPs) is reported by Ma and Powell (2010). A rigorous analysis on solving MDPs more generally by policy iteration with kernel representations is now provided by Farahmand et al. (2016).

1.2. Contributions

Here, our main contribution is to show how the OLSPI algorithm with a polynomial basis for continuous action representation (Bušoniu et al., 2010) can be endowed with a kernel-inspired automatic feature selection method of low computational complexity. Hence, we obtain an automatic OLSPI (AOLSPI) algorithm that preserves the analyzability properties of the LSPI class, yet can be applied in fashion similar to direct adaptive optimal control. Implementing our algorithm requires only a relatively small amount of modifications starting from OLSPI; nonetheless, some critical tuning parameters of the VFA are removed. Hence, practitioners will benefit by easier deployment to actual systems.

In deriving the novel algorithm, we have several side contributions. (1) We start by adding capabilities to the KLSPI from Xu et al. (2007) to work online in the above sense, i. e., under incremental data collection and reduced processing burden. Opposed to Jakab and Csató (2015), Yahyaa and Manderick (2014), we discuss the role of the sparsification scheme to save computational time, based on advances in the field of kernel adaptive filtering. We then (2) obtain a modification of OLSPI's standard temporal difference (TD) update rule, which also allows for a kernel-inspired approach to distribute basis functions for the continuous state and action VFA, without actually applying the kernel trick to OLSPI. To benefit from enhanced information processing nonetheless, (3) the similarity measure of the sparsification process is used to extrapolate learned information to new dictionary elements. Hence, (4) the convergence of the novel algorithm is shown to be eventually similar to that of a well-tuned OLSPI with a fixed set of BFs.

The remainder of this article is organized as follows. First, in Section 2 we recall the main ideas of LSPI and its kernel variant, which leads to the problem statement. The main contribution is given in Section 3: an online LSPI algorithm with automatic tuning capability that is applicable to continuous action space domains. In Section 4 the proposed algorithms are evaluated in a conclusive simulation study and their performance is discussed for a wide range of algorithmic parameters. The article concludes with Section 5, giving an outlook to some future work.

2. Reinforcement learning & (Kernel-based) least-squares policy iteration

We start by briefly recalling the main concepts of reinforcement learning (Sutton and Barto, 1998) in general and least-squares policy iteration (Lagoudakis and Parr, 2003; Bušoniu et al., 2010) in particular, before proceeding to summarizing the kernel-based LSPI variant from (Xu et al., 2007). We then concisely state the problem considered in this article.

2.1. Reinforcement learning

Consider a sequential decision making problem under uncertainty modeled as an MDP, i. e., a tuple $\left(\mathcal{S}, \mathcal{A}, P_a, R, \gamma, s_0\right)$, where $\mathcal{S}$ is a set of possible states with $s_0, s, s' \in \mathcal{S}$ and $\mathcal{A}$ is a set of possible actions $a \in \mathcal{A}$. The probability distribution $P_a(s, s') = P(s' \mid s, a)$ is the model that describes the chance of landing in successor state s' by executing action a, currently being in state s. The fourth element of the MDP is the reward function $R(\cdot)$, which judges the quality of the transition from state s to s', triggered by action a. The scalar discount factor $\gamma \in [0, 1]$ is used to set the focus on short- or long-term rewards. When confronted with an MDP, the goal is to find an optimal policy $\pi^\star : \mathcal{S} \mapsto \mathcal{A}$ that encodes which actions are best to take in a certain state. The corresponding optimal action $a^\star$ is defined as the action that maximizes the return $G = \sum_{t=0}^{\infty} \gamma^t R_{t+1}$, the sum of expected cumulative future discounted reward.

If the dynamics of an MDP are known, i. e., the transition probability $P_a(s, s')$ is known, the optimal policy can be found via planning algorithms, most prominently dynamic programming (DP) (Bellman, 1957; Puterman, 1994; Bertsekas, 1995). The goal of maximizing the return for every possible state leads to the central idea of value- (or critic-)based methods, i. e., maintaining a ranking of all possible states $s \in \mathcal{S}$ of the MDP with the purpose of finding the optimal action $a_t^\star$ in each step t that is expected to lead to the highest ranked successor state s'. This ranking is called the (state) value function $V_\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid s_0 = s\right]$. It is important to note that such a representation can only be created with respect to a policy that determines the state transitions; hence, the subscript π. Solving an MDP refers to finding an optimal policy $\pi^\star$ that maximizes the expected return in all states, $\pi^\star = \arg\max_\pi V_\pi(s), \forall s \in \mathcal{S}$; such $\pi^\star$ always exists (Puterman, 1994). Usage of the state value function $V_\pi(s)$, however, requires knowledge about the transition probabilities $P_a(s, s')$ of the MDP to evaluate possible successor states. Reinforcement learning in turn operates on a trial-and-error basis and does not rely on information about the MDP dynamics. In order to employ the concept of the value function nonetheless, in unknown environments the state value function $V_\pi(s)$ is extended to the state–action value function $Q_\pi(s, a) = \mathbb{E}_\pi\left[G \mid s_0 = s, a_0 = a\right]$ that assigns each state–action pair the expected sum of rewards when starting from state s, taking action a, and henceforth following π. Note that $V_\pi(s) \triangleq Q_\pi(s, \pi(s))$. The state–action space is henceforth denoted $\mathcal{SA} \triangleq \mathcal{S} \times \mathcal{A}$, and a state–action value function $Q : \mathcal{SA} \mapsto \mathbb{R}$ entails a (greedy) policy via

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a). \tag{1}$$

The optimal policy $\pi^\star$ is obtained from the optimal state–action value function $Q^\star(s, a) = \max_\pi Q_\pi(s, a)$. Unfortunately, there are as many state–action value functions $Q_\pi(s, a)$ as there are policies π and value-based RL methods aim to learn the optimal $Q^\star(s, a)$.

2.2. Least-squares policy iteration

Policy iteration (PI) is one particular method to learn $Q^\star(s,a)$. PI tackles the learning problem by starting with some randomly chosen policy and improving it iteratively until convergence to the optimal one. To this end, two steps are alternating. The first is policy evaluation, which refers to computing the state–action value function $Q_\pi(s,a)$ of the current policy. This estimate is then used in the second step, the policy improvement done via (1). The policy evaluation step requires to solve the Bellman equation (Bellman, 1957) of the MDP

$$Q_\pi(s_t,a_t) = \mathbb{E}_\pi\left[R_{t+1} + \gamma Q_\pi(s_{t+1},a_{t+1}) \mid s_t, a_t\right]. \tag{2}$$

In continuous spaces $\mathcal{S}$ or $\mathcal{A}$ that typically occur in physical systems, it is in general not possible to solve the policy evaluation step exactly. In this case, the state–action value function $Q(s,a)$ is commonly approximated as $\hat{Q}(s,a)$ by means of a linear approximation architecture (Bušoniu et al., 2010; Geramifard et al., 2013). To this end, a set $\mathcal{F} = \left\{\phi_i(s,a) : \mathcal{S}\times\mathcal{A} \mapsto \mathbb{R},\ i=1,\dots,N_\phi\right\}$ of features is selected, which consists of N_ϕ state and action dependent BFs $\phi(\cdot,\cdot)$. The approximated value $\hat{Q}$ for a given state–action tuple (s,a) is then computed as a weighted sum of the BFs

$$\hat{Q}(s,a) = \sum_{i=1}^{N_\phi} \phi_i(s,a)\theta_i = \phi(s,a)^\top\theta. \tag{3}$$

Solving (2) approximately by minimizing the approximation error in a least squares sense results in the LSPI algorithm. In its original form (Lagoudakis and Parr, 2003), this algorithm is offline, i. e., it requires a batch of transition data samples of interactions with the environment. Bušoniu et al. (2010) present a variant that processes interactions with the environment on the fly, therefore called OLSPI. Both algorithms build a matrix A and a vector b from subsequent interactions in order to solve the projected Bellman equation by TD learning according to

$$A \leftarrow A + \phi(s,a)\left(\phi(s,a) - \gamma\phi(s',\pi(s'))\right)^\top, \tag{4}$$

$$b \leftarrow b + \phi(s,a)r. \tag{5}$$

LSPI rebuilds these matrices in every iteration from scratch, whereas OLSPI continues to update A and b as long as it interacts with the environment.

In order to use OLSPI over scalar continuous action domains, orthogonal polynomials such as Chebyshev polynomials of the first kind $\Psi_j : [-1,1] \mapsto [-1,1]$ of degree j, $0 \le j \le M$, are used to construct an extended feature vector $\phi(s,a) \in \mathbb{R}^{(M+1)N_\phi}$ as

$$\phi(s,a) = \begin{bmatrix} \phi_{\mathrm{S}}(s)\Psi_0(\bar{a}) \\ \vdots \\ \phi_{\mathrm{S}}(s)\Psi_M(\bar{a}) \end{bmatrix}, \text{ where } \phi_{\mathrm{S}}(s) = \begin{bmatrix} \phi_1(s) \\ \vdots \\ \phi_{N_\phi}(s) \end{bmatrix}. \tag{6}$$

The benefit of working with the extended feature vector (6) is that the approximation over the action space $\mathcal{A}$ is kept separated from that over the state space $\mathcal{S}$. In (6), without loss of generality, the action space $\mathcal{A}$ is scaled to exploit the orthogonality of the Chebyshev polynomials over the set $\bar{\mathcal{A}} \triangleq [-1,1]$, with the elements denoted $\bar{a} \in \bar{\mathcal{A}}$. Thus the policy improvement step (1) becomes tractable: computing (3) for the current state s results in a polynomial expression over $\bar{a}$

$$\Psi(\bar{a}) = c_M\bar{a}^M + \dots + c_1\bar{a} + c_0 \tag{7}$$

which is exactly representable by the coefficients c_j and it remains to compute $\arg\max_{\bar{a}\in\bar{\mathcal{A}}}\Psi(\bar{a})$ to find the greedy step (1) efficiently. Further details on OLSPI with Chebyshev polynomial approximation are skipped for brevity and the reader is referred to the literature (Bušoniu et al., 2010, Ch. 5.3, p. 170ff, and Ch. 5.5, p.177ff). If a vector-valued action space is to be considered, one can simply run several instances of OLSPI in parallel.

2.3. Kernel-based policy iteration

A version of LSPI which exploits the *kernel trick* (Schölkopf and Smola, 2002) to approximate the state–action value function $Q(s,a)$ is presented in Xu et al. (2007). Similar to the linear approximation architecture, the Q function is approximated via a weighted sum of kernel functions, i. e.,

$$\hat{Q}(s,a) = \sum_{i=1}^{N_\mathrm{K}} k_i(s,a)\theta_i = k(s,a)^\top\theta$$

$$\text{with}\quad k_i(s,a) \triangleq \kappa\left((s,a),(s_i,a_i)\right),$$

$$k(s,a) = \left[k_1(s,a),\dots,k_{N_\mathrm{K}}(s,a)\right]^\top. \tag{8}$$

The function $\kappa(x,x') : \mathcal{SA}\times\mathcal{SA} \mapsto \mathbb{R}$ denotes the positive definite symmetric kernel function inducing a reproducing kernel Hilbert space (RKHS), i. e., the feature space $\mathcal{H}$ with inner product $\langle\cdot,\cdot\rangle_\mathcal{H}$ such that

$$\kappa\left(x,x'\right) = \langle\Phi(x),\Phi(x')\rangle_\mathcal{H}. \tag{9}$$

The mapping $\Phi : \mathcal{SA} \mapsto \mathcal{H}$ is the feature map which is implicitly defined by the kernel. The set $\mathcal{D} = \left\{(s_i,a_i) \in \mathcal{SA}, i=1,\dots,N_\mathrm{K}\right\}$ is a dictionary of $N_\mathrm{K} \triangleq |\mathcal{D}|$ collected state–action tuples $x = (s,a)$. Roughly speaking, this set contains a finite number of points representative for the space spanned by $\mathcal{S}\times\mathcal{A}$. We briefly summarize the main steps of the KLSPI algorithm: based on the dictionary, the training data is iterated over in order to recursively solve the projected Bellman equation, leading to an improved policy. Then, the learning agent interacts greedily with its environment and produces new data samples. New samples are added to the dictionary only on per-need basis and the whole process is repeated until some convergence criterion is fulfilled. The advantage of the KLSPI algorithm is two-fold: first, the approximation of the Q function is computed in the RKHS; second, the set $\mathcal{D}$ of representative samples is created in automated fashion. In Xu et al. (2007), this is done via ALD analysis applied to the dictionary state–action tuples (s,a): if a new tuple can be reasonably well represented by a linear combination of the N_K tuples already contained in the dictionary, its addition to the dictionary is not considered justified. Formally, the approximation error is calculated by

$$\delta = K_{**} - K_*^\top K^{-1}K_*, \tag{10}$$

with $K \in \mathbb{R}^{N_\mathrm{K}\times N_\mathrm{K}}$, $K_* \in \mathbb{R}^{N_\mathrm{K}}$, and $K_{**} \in \mathbb{R}$ defined by the Mercer kernel κ, the training data $\mathcal{D}$, and the query input $x' = (s',a')$ as

$$K_{i,j} = \kappa(x_i,x_j), \quad \forall x_i,x_j \in \mathcal{D} \tag{11}$$

$$K_{*i} = \kappa(x_i,x'), \quad \forall x_i \in \mathcal{D} \tag{12}$$

$$K_{**} = \kappa(x',x'). \tag{13}$$

Given a threshold δ_0, the ALD criterion states that x' is already sufficiently well represented by the dictionary if $\delta \le \delta_0$. Accordingly, x' is added to $\mathcal{D}$ if $\delta > \delta_0$. For learning, a TD-like update similar to (4) and (5) is used, employing the vector of kernels $k(s,a)$ in place of the feature vector $\phi(s,a)$:

$$A \leftarrow A + k(s,a)\left(k(s,a) - \gamma k(s',\pi(s'))\right)^\top, \tag{14}$$

$$b \leftarrow b + k(s,a)r. \tag{15}$$

Hence, in the notation above it is clear that the core learning mechanism is quite similar in the LSPI, OLSPI, and KLSPI algorithms.

2.4. Problem statement

With these well-established algorithms in mind, we are now in position to emphasize which parts of the algorithms allow for modifications in order to deploy LSPI more easily to actual robotic systems.

Xu et al. (2007) state that their KLSPI can be used to optimize an existing policy online. This policy, however, is required to feature some level of performance. Due to this initial performance guarantee, the need for additional exploration is avoided. In spite of these assets, the KLSPI algorithms alternates between two main steps: data collection, i. e., greedy interaction with the environment, and subsequent policy improvement. Data is thus processed in batches. Moreover, it is difficult to identify the required performance level of the initial policy. Note that this notion of *online* mechanism contrasts the requirements that typically occur in robotics outlined above.

Problem 1 (*KLSPI for online learning*)**.** Development of an online version of KLSPI, i. e., data should be processed once it becomes available while the per-iteration time must not increase significantly during run-time. ⋄

The OLSPI algorithm from (Bušoniu et al., 2010), in turn, is capable of online processing and continuous action space representations. Yet it should be clear that the choice of features ϕ_i is crucial to obtain good performance in any LSPI algorithm; as pointed out in Geramifard et al. (2013, Ch. 4.5, p. 436), "[…] the choice of the representation can often play a much more significant role in the final performance of the solver than the choice of the algorithm." From a practitioner's point of view, this issue is ubiquitous when having to select basis functions in order to apply approximation-based RL algorithms to robotics, a tuning process that can be tedious. We therefore aim to automate this process.

Problem 2 (*OLSPI with automatic VFA*)**.** Derivation of an OLSPI algorithm that is applicable to continuous state–action spaces and automatically selects suitable features in order to reduce hand-tuning of the VFA, or to obtain a good starting point for subsequent fine-tuning of OLSPI. ⋄

3. Online, continuous-space & automatic LSPI

This section presents our main result, a set of modifications for OLSPI in order to solve Problem 2. To this end, we first provide a solution to Problem 1 and call the resulting algorithm *OKLSPI*.

3.1. Online kernel least-squares policy iteration

The kernel-based RL approaches reviewed in Section 1.1 select data points based on ALD analysis. A first recursive version of KLSPI is presented in Yahyaa and Manderick (2014), however, considering only a discrete state space, using expensive ALD sparsification as well, and it lacks a convergence analysis. We therefore begin by adopting a more efficient sparsification rule.

3.1.1. Sparsification rule

A direct implementation of the ALD criterion (10) requires the inversion of a Gram matrix $K \in \mathbb{R}^{N_\mathrm{K}\times N_\mathrm{K}}$, which results in a basic complexity of $\mathcal{O}(N_\mathrm{K}^3)$ (Rasmussen and Williams, 2006). Clearly, the per-iteration time will increase significantly with the growing dictionary; hence, the matrix inversion should be avoided. One alternative approach is to directly propagate the inverse matrix by recursive updates, similar as done in Jung and Polani (2007), Yahyaa and Manderick (2014). However, the complexity is still $\mathcal{O}(N_\mathrm{K}^2)$ in this case; moreover, learning the inverse results in increased sensitivity w. r. t. the numeric initialization parameters. Recently, other sparsification methods are becoming more mature and well-understood, see e. g. (Honeine, 2015). We therefore propose to adopt another sparsification procedure that inherently is of only linear complexity: the *coherence* criterion introduced in Richard et al. (2009).

The coherence μ of a dictionary $\mathcal{D}$ is defined as

$$\mu = \max_{\substack{i\neq j \\ 1\leq i,j\leq|\mathcal{D}|}} \frac{|\kappa(x_i,x_j)|}{\sqrt{\kappa(x_i,x_i)\kappa(x_j,x_j)}},$$

therefore μ is large if the dictionary contains points x_i and x_j that are very similar in $\mathcal{H}$ as measured by (9). The decision rule whether to include a new sample x' into the dictionary or not is to restrict the coherence of the dictionary below a threshold $0 \leq \mu_0 \leq 1$, i. e., if

$$\max_{x_i\in\mathcal{D}} \frac{|\kappa(x_i,x')|}{\sqrt{\kappa(x_i,x_i)\kappa(x',x')}} < \mu_0, \tag{16}$$

then x' can be added to $\mathcal{D}$. In the following, we assume that a unit-norm kernel function is employed, i. e., a kernel that fulfills $\|\kappa(x,\cdot)\|_\mathcal{H} = 1, \forall x \in \mathcal{SA}$. The most well-known kernel with this property is the Gaussian kernel

$$\kappa(x_i,x_j) = \exp\left(\frac{-1}{2\sigma^2}\|x_i - x_j\|^2\right) \tag{17}$$

and in this case (16) reduces to

$$\max_{x_i\in\mathcal{D}} |\kappa(x_i,x)| < \mu_0. \tag{18}$$

Hence, the complexity of the sparsification rule is reduced to $\mathcal{O}(N_\mathrm{K})$ evaluations of the kernel function and a simple element-wise comparison.

Remark 1 (*Babel criterion*)**.** Instead of the maximum similarity of the data points (i. e., the coherence) as a decision criterion, the cumulative coherence (*Babel* criterion) is sometimes considered for sparsification, see Honeine (2015) for a comparison. In this case, a new data point is included in the dictionary if

$$\sum_{x_i\in\mathcal{D}} |\kappa(x_i,x)| \leq \tilde{\mu}_0.$$

Although of linear complexity as well, for the purpose of *online* RL, this sparsification is not as suitable as the maximum coherence-based diversity measure. The rationale behind will be clarified by means of the simulation study reported in a later part of this article. ⋄

3.1.2. Online dictionary expansion

Rebuilding the matrices A and b in the TD update (14) and (15) from scratch after each interaction is the second shortcoming of KLSPI w. r. t. efficient online data processing. This problem can be circumvented as follows: recall that A is a sum of outer products of the two vectors $u \triangleq k(s,a)$ and $v \triangleq \big(k(s,a) - \gamma k(s',\pi(s'))\big)$. Adding a new feature x to the dictionary $\mathcal{D}$ means to add one dimension u_{n+1} to u and v_{n+1} to v. The resulting outer product $\tilde{u}\tilde{v}^\top$ becomes

$$\tilde{u}\tilde{v}^\top = \begin{bmatrix} u \\ u_{n+1}\end{bmatrix}[v^\top, v_{n+1}] = \begin{bmatrix} uv^\top & uv_{n+1} \\ u_{n+1}v^\top & u_{n+1}v_{n+1}\end{bmatrix}. \tag{19}$$

Thus, only one row and column is added while the other entries remain unaffected. This observation is key to retain the previous values of A and b during the subsequent rank-1 update. To this end, A and b need to be enlarged, e. g., by adding an extra diagonal entry $A_\mathrm{new} \in \mathbb{R}$ to A and an extra entry $b_\mathrm{new} \in \mathbb{R}$ to b as

$$A \leftarrow \mathrm{blkdiag}(A, A_\mathrm{new}) \triangleq \begin{bmatrix} A & 0 \\ 0 & A_\mathrm{new}\end{bmatrix}, \quad b \leftarrow [b^\top, b_\mathrm{new}]^\top, \tag{20}$$

where $\mathrm{blkdiag}(\cdot)$ refers to building the block diagonal matrix. From (14) we then have

$$A \leftarrow \mathrm{blkdiag}(A, A_\mathrm{new}) + \begin{bmatrix} uv^\top & uv_{n+1} \\ u_{n+1}v^\top & u_{n+1}v_{n+1}\end{bmatrix} = \tag{21}$$

$$= \underbrace{\begin{bmatrix} A & 0 \\ 0 & 0\end{bmatrix} + \begin{bmatrix} uv^\top & 0 \\ 0 & 0\end{bmatrix}}_{(\star)} + \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & A_\mathrm{new}\end{bmatrix} + \begin{bmatrix} 0 & uv_{n+1} \\ u_{n+1}v^\top & u_{n+1}v_{n+1}\end{bmatrix}}_{(\star\star)}. \tag{22}$$

Conceptually, the resulting TD update (21) can be conceived of as the decomposition (22): it corresponds to a TD step ($\star$) as if the dictionary had not been modified, and the additional part ($\star\star$) is a TD step for the new point starting from A_new. Obviously, the values of A and b computed during prior iterations remain unchanged and therefore can be re-used directly after the dictionary is expanded. Further, it is always possible to choose $A_\mathrm{new} = 0$ and $b_\mathrm{new} = 0$; however, a better method to obtain A_new and b_new is proposed later in Section 3.3.

Table 2
Online kernel least-squares policy iteration with coherence sparsification and efficient dictionary expansion.

Algorithm 1 Online KLSPI (OKLSPI)

```
1  Input: $\kappa(\cdot,\cdot)$ — unit-norm (Mercer) kernel function
          $0 \le \gamma < 1$ — discount factor
          $0 < \mu_0 \le 1$ — coherence threshold
          $\{0 < \epsilon_t < 1\}_{t=0}^{\infty}$ — exploration schedule
          $\beta > 0$ — small positive constant
          $K_\theta \in \mathbb{N}$ — policy improvement interval
          $x_0 = (s_0, a_0)$ — initial state/action tuple
2  $\mathcal{D} \leftarrow (s_0, a_0)$
3  $A, b, \theta, l \leftarrow 0$
4  $s_t \leftarrow$ initial state $s_0$
5  for $t = 0, 1, 2, 3, \ldots$ do
6      $a_t \leftarrow \begin{cases} \text{unif. rand. action in } \mathcal{U} & \textbf{if } \text{rand}([0,1]) \le \epsilon_t \\ \pi(s_t) & \textbf{else} \end{cases}$
7      apply $a_t$, measure $s_{t+1}, r_{t+1}$
8      $\mu = \max\left(\{|\kappa(x_i, (s_t, a_t))| : x_i \in \mathcal{D}\}\right)$
9      if $\mu < \mu_0$ then
10         $\mathcal{D} \leftarrow \mathcal{D} \cup (s_t, a_t)$
11         $A \leftarrow \text{blkdiag}(A, A_{\text{new}}), \quad b \leftarrow [b^\top, b_{\text{new}}]^\top$
               with $A_{\text{new}}$ and $b_{\text{new}}$ according to Eq. (30)
12     end if
13     $A \leftarrow A + k(s_t, a_t)\left(k(s_t, a_t) - \gamma k(s_{t+1}, \pi(s_{t+1}))\right)^\top$
14     $b \leftarrow b + k(s_t, a_t) r_{t+1}$
15     if $t = (l+1)K_\theta$ then
16         $\theta \leftarrow A^{-1} b$
17         $\pi = \underset{a \in \mathcal{A}}{\text{argmax}} \sum_{i=1}^{|\mathcal{D}|} \theta_i \kappa((s_i, a_i), (s, a))$
18         $l \leftarrow l + 1$
19     end if
20 end for
```

With these measures, we obtain the OKLSPI algorithm in Table 2. Clearly, the algorithm contains basic building blocks of both the KLSPI and OKLSPI algorithms. Lines 1–4 initialize the algorithm and the control loop is set up in line 5. In line 6, either a random exploratory or the exploitative action is chosen via the standard ϵ-greedy mechanism. Line 7 describes the interaction with the environment, i. e., the application of action a_t and the measurement of the successor state s_{t+1} and corresponding reward r_{t+1}. The lines 8–9 constitute the coherence sparsification criterion, and if needed the dictionary expansion is done in lines 10–11. The remaining lines 13–19 constitute the standard kernelized TD update. For the practitioner, we would like to emphasize that the policy improvement step in line 17 is of conceptual nature only: it suffices to perform the calculation in line 6 when choosing an exploitative action.

3.2. *Automated online least-squares policy iteration*

Albeit online capability, the proposed OKLSPI algorithm only works with discrete action sets, a shortcoming of major concern for application on robotic devices. Recall from (6) that the OLSPI algorithm handles continuous action spaces by incorporating Chebyshev polynomials of the first kind into an extended feature vector $\phi(s, a)$. However, an analogous extension of the kernel-based LSPI algorithm is not yet known because the similarity of the features in the RKHS is computed implicitly using the kernel trick. In principle, one could analogously construct a kernel for continuous $\mathcal{SA}$ by composition with a suitable orthogonal polynomial kernel (Pan et al., 2012). Nonetheless, the policy improvement step (1) could not be solved exactly anymore by means of a polynomial (7) because this would require to explicitly consider the feature map Φ of (9). This is, however, contrary to the key idea of kernel methods that one does not need to know an explicit form of the feature map but only implicitly define it via (9). Therefore, we propose to rather combine the automated feature selection of the kernel-based approach with the OLSPI algorithm, which allows to use continuous space approximations. To this end, we automate the approximation over the state space by means of kernels but continue to construct the action space approximation using orthogonal polynomials. The resulting algorithm is termed *automated online least-squares policy iteration (AOLSPI)* and provides a solution to Problem 2.

First, we need to build a dictionary over the state space $\mathcal{S}$ only, with an appropriate sparsification rule. To this end, we may simply adopt the previous approach, i. e., a dictionary $\mathcal{D}_\text{S}$ with sparsification criterion (18). We can now replace the basis function vector $\phi_\text{S}(s)$ in the extended feature construction (6) by a vector $k_\text{S}(s)$

$$k_\text{S}(s) = [k_1(s), \ldots, k_{N_\text{S}}(s)]^\top, \; k_i(s) = \kappa(s, s_i), \; s_i \in \mathcal{D}_\text{S} \subset \mathcal{S}, \tag{23}$$

with a unit-norm kernel function $\kappa(\cdot,\cdot)$ and the number of dictionary elements $N_\text{S} = |\mathcal{D}_\text{S}|$. The corresponding feature vector $\hat{\phi}$ is now given by

$$\hat{\phi}(s, a) = \left[k_\text{S}^\top(s)\Psi_0(a), \; \ldots, \; k_\text{S}^\top(s)\Psi_M(a)\right]^\top. \tag{24}$$

Next, the key question is how the growing dictionary can be handled in OLSPI. As evident from (24), the feature vector $\hat{\phi}(s, a)$ now consists of stacked state-dependent vectors of BFs $k_\text{S}(s)$, which are multiplied with Chebyshev polynomials of increasing, but maximum order M. Consequently, a new element in the dictionary $\mathcal{D}_\text{S}$ leads to an increase of the feature vector size by $M+1$ elements. Therefore, the adjustment of matrix A and vector b after a dictionary update needs to be carried out differently than in the case of OKLSPI.

Consider how the corresponding TD update $\Delta(A)$ of matrix A is now calculated using (4):

$$\Delta(A) = \hat{\phi}(s, a)\left(\hat{\phi}(s, a) - \gamma\hat{\phi}(s', \pi(s'))\right)^\top$$
$$= \begin{bmatrix} k_\text{S}(s)\Psi_0(a) \\ \vdots \\ k_\text{S}(s)\Psi_M(a) \end{bmatrix} \begin{bmatrix} k_\text{S}(s)\Psi_0(a) - \gamma k_\text{S}(s')\Psi_0(\pi(s')) \\ \vdots \\ k_\text{S}(s)\Psi_M(a) - \gamma k_\text{S}(s')\Psi_M(\pi(s')) \end{bmatrix}^\top.$$

By examining the element of the first row and first column of $\Delta(A)$ exemplarily, it can be observed that $\Delta(A)$ consists of blocks, each containing a sum of outer products of the state dependent BFs vector. For example, the first block yields

$$\Delta(A)_{(1,1)} = \; k_\text{S}(s)k_\text{S}^\top(s)\Psi_0(a)^2 - \gamma k_\text{S}(s)k_\text{S}^\top(s')\Psi_0(a)\Psi_0(\pi(s')).$$

Similarly, the other blocks differ only by the values of the Chebyshev polynomials that are multiplied to the two outer products $k_\text{S}(s)k_\text{S}^\top(s)$ and $k_\text{S}(s)k_\text{S}^\top(s')$. At this point, the reasoning about outer products of growing vectors (19) applies, i. e., the resulting matrix of the outer product of the vector of state-dependent BFs needs to be expanded by an extra row and an extra column. Note that this applies to all of the blocks in $\Delta(A)$. By analogous derivation for the TD update of b it is immediate that adding an element at every $(N_\text{S}+1)$th index is required. Formally, we obtain the expansion

$$A_{t+1} = \left[\begin{array}{c|c|c} A_{t+1\,(1,1)} & \cdots & A_{t+1\,(1,M+1)} \\ \hline \vdots & \ddots & \vdots \\ \hline A_{t+1\,(M+1,1)} & \cdots & A_{t+1\,(M+1,M+1)} \end{array}\right], \tag{25}$$

where each block is enlarged as

$$\begin{aligned} A_{t+1\,(i,j)} &= \text{blkdiag}(A_{t\,(i,j)}, \; A_{t,\text{new}\,(i,j)}), \\ A_{t\,(i,j)} &= k_\text{S}(s)k_\text{S}^\top(s)\,\Psi_{i-1}(a)\Psi_{j-1}(a) \\ &\quad - \gamma k_\text{S}(s)k_\text{S}^\top(s')\,\Psi_{i-1}(a)\Psi_{j-1}(\pi(s')), \end{aligned} \tag{26}$$

and the block-partitioned vector update

$$b_{t+1} = \left[\begin{array}{c} b_{t+1\,(1)} \\ \hline \vdots \\ \hline b_{t+1\,(M+1)} \end{array}\right] = \left[\begin{array}{c} b_{t\,(1)} \\ b_{t,\text{new}\,(1)} \\ \hline \vdots \\ \hline b_{t\,(M+1)} \\ b_{t,\text{new}\,(M+1)} \end{array}\right] \tag{27}$$

Table 3
OLSPI with automatic basis function selection (AOLSPI). Bold line numbers indicate key differences w. r. t. OLSPI from (Bušoniu et al., 2010).

Algorithm 2 Automated OLSPI (AOLSPI)

```
**1** Input: $\kappa(\cdot,\cdot)$ — unit-norm kernel function
          $M \in \mathbb{N}$ — small integer number, highest degree of Chebyshev polynomials
          $0 \le \gamma < 1$ — discount factor
          $0 < \mu_0 \le 1$ — coherence threshold
          $\{0 < \varepsilon_t < 1\}_{t=0}^{\infty}$ — exploration schedule
          $\beta > 0$ — small positive constant
          $K_\theta \in \mathbb{N}$ — policy improvement interval
          $x_0 = (s_0, a_0)$ — initial state/action tuple
**2** $\mathcal{D}_S \leftarrow s_0$
3 $A, b, \theta, l \leftarrow 0$
4 $s_t \leftarrow$ initial state $s_0$
5 for $t = 0, 1, 2, 3, \dots$ do
6     $a_t \leftarrow \begin{cases} \text{unif. rand. action in } \mathcal{U} & \textbf{if } \text{rand}([0,1]) \le \varepsilon_t \\ \pi(s_t) & \textbf{else} \end{cases}$
7     apply $a_t$, measure $s_{t+1}, r_{t+1}$
**8**     $\mu = \max\left(\{|\kappa(s_i, s_t)| : s_i \in \mathcal{D}_S\}\right)$
**9**     if $\mu < \mu_0$ then
**10**         $\mathcal{D}_S \leftarrow \mathcal{D}_S \cup (s_t)$
**11**         $A, b \leftarrow$ expansion according to Eq. (25) – Eq. (28)
**12**     end if
13     $A \leftarrow A + \hat{\phi}(s_t, a_t)\left(\hat{\phi}(s_t, a_t) - \gamma\hat{\phi}(s_{t+1}, \pi(s_{t+1}))\right)^\top$
14     $b = b + \hat{\phi}(s_t, a_t) r_{t+1}$
15     if $t = (l+1)K_\theta$ then
16         $\theta \leftarrow A^{-1} b$
17         $\pi = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \hat{\phi}^\top(s_t, a_t)\theta$
18         $l \leftarrow l + 1$
19     end if
20 end for
```

with

$$b_{t\,(i)} = k_S(s)\Psi_i(a). \tag{28}$$

Again, $A_{t,\text{new}\,(i,j)} = 0$ and $b_{t,\text{new}\,(i)} = 0$ are always possible choices and we give a preferred way to initialize the new entries in the next section.

The resulting automated online least-squares policy iteration (AOLSPI) algorithm is summarized in Table 3. Compared to the OLSPI algorithm reviewed in Section 2, only the lines 8–12 have to be added. It is therefore straightforward to enhance existing OLSPI implementations in order to realize the automatic VFA capability. Note that, as opposed to OKLSPI, the kernel activation in lines 8–10 only depends on the system state s, whereas the dependency of the extended feature vector $\hat{\phi}$ on the action a is captured via the Chebyshev basis as in OLSPI. Therefore, the implementation of policy improvement remains tractable by means of the polynomial (7).

3.3. Similarity-based information extrapolation in TD update

Next, we examine how the online algorithms presented above process information after the dictionary expansion step. In a single TD update step, the algorithms in this article spread information over multiple elements of A and b, based on the similarity of the dictionary points w. r. t. the current and successor states, see (14) and (15) with (8), respectively (26) and (28) with (23). This mechanism is essential for learning, but partly disabled in the case of AOLSPI and OKLSPI: a new BF that was added to the dictionary some time after the learning process had started clearly missed out on the information that had been spread in the previous interactions with the environment. Taking $A_\text{new} = 0$ and $b_\text{new} = 0$ assumes that there is not yet any information

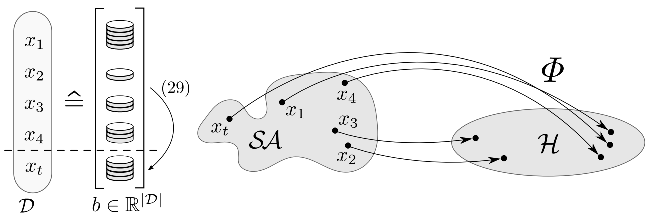


Fig. 1. Visualization of the proposed similarity-based information extrapolation (29) for the TD update of b: according to (15), in each iteration, every entry of the vector b receives a certain amount of the reward r determined by the kernel activation. Therefore, b accumulates the rewards corresponding to each element $x \in \mathcal{D} \subset \mathcal{SA}$. When the dictionary is expanded by a new element x_t, b_new can in consequence approximately be initialized with a weighted average of the collected rewards of the most similar dictionary points. Note that similarity is considered in the feature space $\mathcal{H}$: in the depicted example, x_1 and x_4 contribute most.

about the corresponding part of the state space—after all, it is a new point in the dictionary. By the subsequent interactions of the system with its environment, the information gap of the new BF will be closed asymptotically.

The dependency of the TD step on the similarity of the current and next states w. r. t. the dictionary elements implies, however, that regions of matrix A and vector b which correspond to similar BFs should also have similar values in A and b. Hence, the similarity to the existing grid points as measured by the kernel function can be used to extrapolate entries of A and b to a new dictionary element. This idea is visualized in Fig. 1. While in this section, the formulas are derived to perform an approximative initialization, the numerical example in a later section will demonstrate its utility. Since the structure of A and b is dependent on the algorithm, the corresponding extrapolation rules are different and the OKLSPI-specific extrapolation is introduced first and then ported to AOLSPI.

3.3.1. OKLSPI

For the derivation of the basic extrapolation rule, let us revisit the TD update rule of b given in (15), which is repeated here for the reader's convenience:

$$b \leftarrow b + k(s,a)r.$$

Observe that the elements of b are updated by a fraction of the received reward r as determined by the similarity of the current sample (s,a) with the elements of the dictionary. Grid points similar to each other will thus feature approximately the same values b_i. Thus, we can safely assume that the true value of b_new of a new BF should be of same magnitude as the values of b corresponding to the most similar dictionary points. The value of the new element b_new can therefore be obtained by extrapolation of the existing elements of b weighted by the corresponding similarity, i. e.,

$$b_\text{new} = \frac{\sum_{l=1}^{|\mathcal{D}|} k_l(s,a) b_l}{\sum_{l=1}^{|\mathcal{D}|} k_l(s,a)}. \tag{29}$$

Extrapolating new elements of A is not as straightforward. Let us write out the TD update rule of A from (14) in expanded form:

$$A \leftarrow A + k(s,a)k(s,a)^\top - \gamma k(s,a)k(s',\pi(s'))^\top.$$

The TD update of A consists of a subtraction of two outer products $k(s,a)k(s,a)^\top$ and $k(s,a)k(s',\pi(s'))^\top$. Recall that the coherence-based sparsification rule entails that the elements of the dictionary are dissimilar to a certain extent. Consequently, the first outer product mainly updates elements on the diagonal of A. If the samples (s,a) and $(s',\pi(s'))$ differ, the second outer product mainly affects off-diagonal elements. To extrapolate these elements, knowledge about the previous evolution of the policy would be required. In summary, we can assume that the update of the on-diagonal elements still mainly depends on the kernel vector $k(s,a)$. Hence, an initialization for the new diagonal element A_{new} of the expanded matrix is obtained by a weighted average over the other diagonal elements as

$$A_{\text{new}} = \frac{\sum_{l=1}^{|\mathcal{D}|} k_l(s,a) A_{ll}}{\sum_{l=1}^{|\mathcal{D}|} k_l(s,a)}.$$

The strength of the extrapolation can be varied by actively restricting the number of considered grid points to a set $\tilde{\mathcal{D}} \subseteq \mathcal{D}$, yielding

$$A_{\text{new}} = \frac{\sum_{l=1}^{|\tilde{\mathcal{D}}|} k_l(s,a) A_{ll}}{\sum_{l=1}^{|\tilde{\mathcal{D}}|} k_l(s,a)}, \quad b_{\text{new}} = \frac{\sum_{l=1}^{|\tilde{\mathcal{D}}|} k_l(s,a) b_l}{\sum_{l=1}^{|\tilde{\mathcal{D}}|} k_l(s,a)}. \tag{30}$$

The set $\tilde{\mathcal{D}}$ can be taken, for example, by ranking the similarity to the new BF and selecting only a percentage $p_{\text{e}} \leq 1$ of most similar points. We call this approach *trust radius* in the following. The complete dictionary $\tilde{\mathcal{D}} = \mathcal{D}$ is used for $p_{\text{e}} = 1$; for $\tilde{\mathcal{D}} = \emptyset$ in turn, the conservative initialization of the new elements with zero $A_{\text{new}} = 0$ and $b_{\text{new}} = 0$ is recovered.

3.3.2. AOLSPI

For the AOLSPI algorithm of Table 3, we adopt the extrapolation method of OKLSPI. It is essentially the same mechanism, yet applied separately to the segments of A and b. When enlarging the vector b as (27), the newly added entry $b_{t+1,\text{new}\,(i)}$ in every segment $b_{t+1\,(i)}, i = 1 \dots M+1$, is an average of the other elements of the ith block segment of b, weighted by the similarity of the corresponding BF grid point to the grid point of the new BF, i. e.,

$$b_{t+1,\text{new}\,(i)} = \frac{\sum_{l=1}^{|\tilde{\mathcal{D}}|} k_l(s) b_{t\,(i)(l)}}{\sum_{l=1}^{|\tilde{\mathcal{D}}|} k_l(s)}. \tag{31}$$

The values of A are extrapolated again in a more conservative way by considering only the block elements on the diagonal. Within these blocks $A_{t+1\,(i,i)}$, the Chebyshev polynomials are equal. Hence, the two outer products are scaled by the same value and (26) simplifies to

$$A_{t\,(i,i)} = k_{\text{S}}(s) k_{\text{S}}^{\top}(s) \Psi_i^2(a) - \gamma_{\text{S}} k_{\text{S}}(s) k_{\text{S}}^{\top}(s') \Psi_i(\pi(s')) \Psi_i(a).$$

Now as in the case of OKLSPI, within the corresponding block, the first outer product $k_{\text{S}}(s)k_{\text{S}}^{\top}(s)$ updates mainly on-diagonal elements. The other outer product $k_{\text{S}}(s)k_{\text{S}}^{\top}(s')$ further updates on-diagonal elements if s and s' are similar; otherwise, off-diagonal elements are updated depending on the policy π. The interpolation is therefore again restricted to the diagonal elements of the related block and the initialization of the new element is correspondingly

$$A_{t,\text{new}\,(i,j)} = \frac{\sum_{l=1}^{|\tilde{\mathcal{D}}|} k_l(s) A_{t\,(i,j)(l,l)}}{\sum_{l=1}^{|\tilde{\mathcal{D}}|} k_l(s)}. \tag{32}$$

The number of used grid points can be selected according to a trust radius approach as in (30).

3.4. Convergence analysis

In this section, we briefly comment on the convergence of the novel algorithms. Recall that AOLSPI automates the process of selecting basis functions for OLSPI; further it is clear that the VFA plays a crucial role in the performance of OLSPI.

Remark 2 (*Performance guarantees of online LSPI*)**.** Unfortunately, to the best of the authors' knowledge, even the asymptotic properties of OLSPI with a fixed set of BFs are not yet completely understood, cf. (Buşoniu et al., 2012, Ch. 3.6.1, p. 97). The basic reason behind is that the policy improvement step in OLSPI is taken according to only an approximation of the value function. In consequence, the policy evaluation error may become large and the performance assertions of the basic LSPI (Lagoudakis and Parr, 2003) do not necessarily carry over to the online case (Buşoniu et al., 2010). ⋄

Concerning the approximation architecture, however, Ma and Powell are able to show (Ma and Powell, 2009; Powell and Ma, 2011) that under certain conditions, approximate policy iteration with Chebyshev polynomials converges in the mean. Thus, our effort is to show that the modifications introduced in this article do at least preserve the convergence properties of the prior algorithms. First, observe that, as proven by Richard et al. (2009, Prop. 2), the size of the feature vector ϕ converges to a fixed size at some time T, namely when the state space is completely covered with BFs as governed by the sparsification procedure and the fixed threshold μ_0. Henceforth, in all subsequent samples $t > T$, AOLSPI reduces to OLSPI as will be shown next. In the first place, the samples collected during $0 \leq t \leq T$ only contributed partly to the TD update (4) and (5) of A and b. This is because the associated BFs had not been part of the dictionary yet, hence the corresponding entries could not be updated. However, after convergence of the dictionary, i. e., considering $t > T$, the feature vector basis is now fixed. We may hence think of the incomplete updates during $0 \leq T$ as some corrupted feature vectors ϕ_{c} affecting A and b. In the limit, the learning mechanism described by (4) and (5) becomes

$$A = \underbrace{\lim_{N\to\infty} \frac{1}{N} \sum_{i=0}^{T} \phi_{\text{c}}(s_i,a_i) \left(\phi_{\text{c}}(s_i,a_i) - \gamma \phi_{\text{c}}(s_i', \pi(s_i'))\right)^{\top}}_{\to 0}$$
$$+ \lim_{N\to\infty} \frac{1}{N} \sum_{i=T+1}^{N} \phi(s_i,a_i) \left(\phi(s_i,a_i) - \gamma \phi(s_i', \pi(s_i'))\right)^{\top},$$
$$b = \underbrace{\lim_{N\to\infty} \frac{1}{N} \sum_{i=0}^{T} \phi(s_i,a_i) r_i}_{\to 0} + \lim_{N\to\infty} \frac{1}{N} \sum_{i=T+1}^{N} \phi(s_i,a_i) r_i.$$

The limit in the first summand in both expressions exists and approaches zero as $N \to \infty$ because the sum of bounded matrices is bounded. By substitution of $i = T+1$ with $i = 0$ and reformulation, the remaining solution in the limit approaches that of the OLSPI algorithm

$$A^{\star} = \lim_{N\to\infty} \frac{1}{N} \sum_{i=0}^{N} \phi(s_i,a_i) \left(\phi(s_i,a_i) - \gamma \phi(s_i', \pi(s_i'))\right)^{\top}$$
$$b^{\star} = \lim_{N\to\infty} \frac{1}{N} \sum_{i=0}^{N} \phi(s_i,a_i) r_i, \qquad \theta^{\star} = (A^{\star})^{-1} b^{\star}.$$

In principle, (sub-)optimality of $\theta^{\star}$ could be established according to Lagoudakis and Parr (2003, Th 7.1), i. e., the error norm of the performance of the policies w. r. t. the optimal performance is in the limit bounded by some constant, subject to the restrictions of Remark 2 concerning online LSPI.

In summary, it is shown that the limit convergence behavior is independent of the specific dictionary sparsification method as long as $|\mathcal{D}|$ is finite, and that further the dictionary expansion and data extrapolation scheme introduced above do not void the general performance behavior of OLSPI. On the contrary, our simulation studies reported in the next section suggest that the speed of convergence may be considerably improved using AOLSPI and the scheme from Section 3.3.

Analogously to the previous line of argumentation, the convergence of the OKLSPI algorithm could be analyzed given the technical assumptions in Ma and Powell (2010), Powell and Ma (2011).

3.5. Complexity analysis and optimized implementation

Let us briefly argue that the *additional* computational complexity w. r. t. OLSPI induced by our modifications is linear in the number of dictionary elements $n = |\mathcal{D}|$, i. e., an additional $\mathcal{O}(n)$ operations must be performed to implement either of the OKLSPI or AOLSPI algorithms. Consider OLSPI as starting point, as it is the underlying online algorithm in both cases. For the AOLSPI algorithm, the only additional operations are those of lines 8–12 in Table 3. Note that $k_{\text{S}}(s_t)$ must be computed to obtain $\hat{\phi}(s,a)$ of (24) one way or the other.

Summarizing the remaining elementary scalar operations, we have an additional computational complexity of $\mathcal{O}(n)$ operations. A similar line of reasoning is applicable to OKLSPI: in terms of complexity, we can think of Table 2 as an instance of OLSPI with a discrete action space. Again, counting the remaining operations to grow the dictionary corresponding to lines 8–12 in Table 2, the added complexity is $\mathcal{O}(n)$.

For implementation, an optimized version of the basic LSTD-Q algorithm is given in Lagoudakis and Parr (2003, Fig. 6), analogously for KLSTD in Jakab and Csató (2015), that avoids the $\mathcal{O}(n^3)$ inversion of A by means of the matrix-inversion lemma. Our algorithms are amenable to such an approach as well: recall that the dictionary expansion and information extrapolation steps exploit the prevailing diagonal entries in the matrix structure. Therefore, similar steps could be applied when propagating the inverse matrix. Our simulation studies indicate, however, that the performance of the resulting algorithm is much more sensitive w. r. t. the numeric initialization parameter needed to avoid an ill-posed system. We therefore refrain from discussing the details here and suffice it to say that the approximations concerning the block matrix structure with single block diagonal elements remain unaffected by learning the inverse matrix directly. Thus, an optimized implementation of AOLSPI or OKLSPI based on Sherman–Morrison is feasible in principle, albeit at the cost of a more sensitive parameter set.

4. Simulation study example

Due to the limitations of value-based RL algorithms discussed in the introduction, policy search algorithms may be a more suitable choice for example in high-dimensional robotic learning control problems. If, however, an LSPI approach is appropriate for the control problem at hand, the algorithms proposed in this article constitute an online value-based approach capable of efficient, automatic VFA. Therefore, the task of having to explicitly distribute basis functions in a multidimensional space is avoided. While it is not expected that the presented online algorithms generally outperform their hand-tuned counterparts, a similar level of performance should be attained as by OLSPI in a well-tuned setting. In order to exemplify the two novel algorithms and evaluate their performance, we consider two standard LSPI benchmark scenarios and compare the results to those obtained with the established LSPI algorithms using well-tuned parameters.

4.1. OKLSPI and the car on the hill problem

We will first illustrate how the OKLSPI algorithm of Table 2 indeed solves Problem 1. In other words, it is demonstrated that the online dictionary expansion and sparsification measures proposed in Sections 3.1 and 3.3 are adequate. To this end, let us consider the car on the hill problem, a standard benchmark in approximate RL that can be found in Bušoniu et al. (2010) and the references therein. In this task, a point mass (the car) should climb a hill by applying a horizontal force; however, the force is not strong enough to climb the hill directly. Therefore, the car needs to swing back and forth first in order to pump energy in the system. Normalizing quantities to their base SI units, the hill is modeled as a function $H(p)$, where $p \in [-1, 1]$ denotes the horizontal position of the car:

$$H(p) = \begin{cases} p^2 + p, & \text{if } p < 0, \\ \frac{p}{\sqrt{1+5p^2}} & \text{otherwise.} \end{cases}$$

With the discrete control input $u \in \{-4, 4\}$, $g = 9.81$ the gravitational constant, and $\dot{p} \in [-3, 3]$ the velocity of the car, the continuous-time dynamics are given by Bušoniu et al. (2010, p. 160)

$$\ddot{p} = \frac{1}{1 + \left(\frac{\mathrm{d}H(p)}{\mathrm{d}p}\right)^2} \left(u - g\frac{\mathrm{d}H(p)}{\mathrm{d}p} - \dot{p}^2 \frac{\mathrm{d}H(p)}{\mathrm{d}p} \frac{\mathrm{d}^2 H(p)}{\mathrm{d}^2 p} \right)$$

Table 4
OKLSPI parameter settings for the car on the hill problem.

Parameter	Value
Discount factor γ	0.97
Exploration factor ε	0.95
Minimum exploration $\varepsilon_{\min}$	0.05
Number of BF ϕ	dynamic
Kernel function κ	(34)
RBF variance Σ	$\mathrm{diag}(0.1, 0.2, 0.1)$
Update interval K_θ	5

With the reward function

$$R = \begin{cases} -1, & \text{if } p < -1 \text{ or } |\dot{p}| > 3 \\ 1, & \text{if } p \geq 1 \text{ and } |\dot{p}| \leq 3 \\ 0, & \text{otherwise,} \end{cases}$$

the cost landscape as well as optimal Q-functions are discontinuous and therefore hard to approximate as shown in Bušoniu et al. (2010, Ch. 4.5.4). The experiments reported next were conducted with MATLAB R2018a, using the `ode45` solver for numeric integration and a sample time of $t_S = 0.1$ s for discretization.

Let us first give an intuition how the sparsification criterion affects the dictionary growth and the computation times. In order to compare the behavior of OKLSPI with coherence sparsification according to Section 3.1.1 to that of ALD sparsification, we also implemented Algorithm 1 with lines 8–9 replaced by the ALD criterion given from (10)–(13). Next, a simple parameter sweep over 99 learning runs with OKLSPI is conducted for the threshold parameters δ_0 of ALD chosen in a logarithmic scale between $[10^{-5}, 10^1]$, respectively μ_0 of coherence chosen linearly in the interval $[0.01, 0.99]$. The parameters of the OKLSPI algorithm are set according to Table 4 unless stated differently. Each simulation run consists of 75 trials and during each trial of 2 s, the algorithm is granted $2/0.1 = 20$ interactions with the system before being reset to a random admissible initial state. Being an online algorithm, it is essential to use sufficient exploration during the data generation and we simply use the ε-greedy mechanism. Thereby, the exploration probability in time step t is governed by

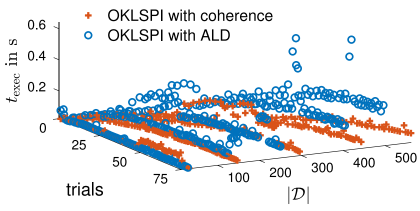
$$\varepsilon_t = \max\left(\varepsilon\left(1 - \frac{t_S \cdot t}{0.75 t_{\max}}\right), \varepsilon_{\min}\right), \tag{33}$$

where $t_{\max} = 2$ s is the duration of a single learning trial. We use a Gaussian kernel function

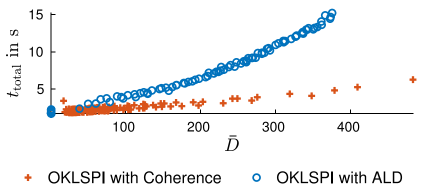
$$k_i(x) = \exp\left(-\frac{1}{2}(x - x_i)^\top \Sigma^{-1}(x - x_i)\right). \tag{34}$$

In order to evaluate the influence of the sparsification criterion on the execution times of the algorithm, we used a straightforward implementation to approximately measure the calculation times t_{exec} for each trial. The experiment was done on a Linux machine with the processor set to a constant CPU frequency of 1.8 GHz. The results of this experiment are shown in Fig. 2.

Fig. 2(a) shows how the dictionary size $|\mathcal{D}|$ grows with increasing trials; the depicted runs were obtained by choosing values of δ_0 and μ_0 such that the amount of kernel functions in the dictionary is in the same order of magnitude for both sparsification methods. It can be seen that the execution times increase notably when ALD is used, particularly if the dictionary size is in the magnitude of hundreds. The outliers in the plot are presumably due to the imprecise method of measuring t_{exec}. In order to show the trend more clearly, Fig. 2(b) depicts the plot of $t_{\text{total}} = \sum_{i=1}^{75} t_{\text{exec},i}$ over the average dictionary sizes $\bar{D} = \frac{1}{75}\sum_{i=1}^{75} |\mathcal{D}_i|$ for all the 99 runs. The measured results are consistent with the theoretical discussion in Section 3.1.1 concerning the complexity of the sparsification criteria. These results illustrate that the per-iteration time remains reasonable using the proposed OKLSPI algorithm with coherence sparsification and K_θ high enough (for the fully optimistic case $K_\theta = 1$, the algorithm performs more expensive policy improvement steps in each iteration).



(a) Execution times t_{exec} for 75 trials of OKLSPI. The depicted runs were obtained (from left to right) for $\delta_0 \in [0.99, 0.1, 10^{-2}, 10^{-4}, 10^{-5}]$ (ALD) and $\mu_0 \in [0.1, 0.75, 0.9, 0.95, 0.97]$ (coherence).



(b) Total execution times t_{total} per trial over the average number of dictionary elements created during the 99 runs with different sparsification thresholds.

Fig. 2. Comparison of the execution times of OKLSPI in the car on the hill problem. It can be seen that the times increase with increasing dictionary size $|\mathcal{D}|$ and that the increase is much stronger when using ALD sparsification. Therefore, the coherence criterion is more suitable for *online* reinforcement learning control with automatic VFA.

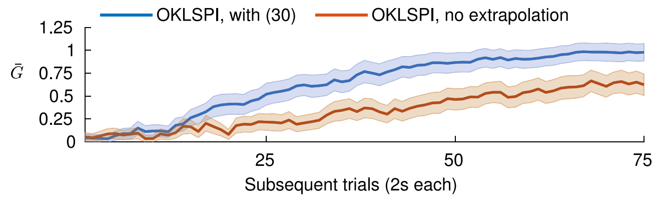


Fig. 3. Performance of OKLSPI in the car on the hill problem with $\mu_0 = 0.9$, corresponding to an average dictionary size of $|\bar{D}| \approx 240$. The figure depicts the mean score $\bar{G}$ according to (35) over the 90 runs (thick lines) and the corresponding 95% confidence intervals (shaded areas). The TD update information extrapolation after insertion of a new dictionary element is according to Section 3.3 with the trust radius $p_e = 1$.

In order to investigate the performance of the proposed OKLSPI algorithm, the following procedure is used. The algorithm is evaluated over $N_{\text{eval}} = 90$ independent runs, where each run consists of 75 trials each starting from a random initial state and given $t_{\max}/t_S = 20$ interactions with the system for learning. To assess the quality of the policy over time, after each trial, the average return is calculated obtained when following the current policy without exploration for three initial states $\mathcal{S}_0 = \{[-0.8, 0]^\top, [-0.4, 0]^\top, [0, 0]^\top\}$, i. e.,

$$\bar{G} = \frac{1}{|\mathcal{S}_0|} \sum_{i=1}^{|\mathcal{S}_0|} \sum_{j=1}^{N_{\text{test}}} \gamma^j R_j. \tag{35}$$

The second and third initial states do not allow to drive the car up the hill just by applying the maximum input but require the policy to swing back and forth.

A plot of a representative learning curve is shown in Fig. 3 for $\mu_0 = 0.9$ and similar plots are obtained for a wide range of the sparsification parameter μ_0. The utility of the TD extrapolation scheme according to (30) becomes evident as well, although its effect varies with the number of useful similar dictionary elements, i. e., it depends on μ_0. This example demonstrates how straightforward it is to implement and tune the algorithm, opposed to alternative value-based approaches that require more tedious tuning of the approximation architecture such as fuzzy Q-iteration, cf. (Bušoniu et al., 2010, Ch. 4.5.4).

Finally, let us remark that we refrain from trying to compare the performance to that of offline KLSPI. It is not clear how to construct a meaningful assessment: being an offline algorithm, KLSPI was not designed to operate under online conditions and one would need to find an unbiased test scenario. As KLSPI re-iterates over its growing training data set from the beginning in each iteration, the number of direct interactions with the test system would somehow have to be restricted in order to enforce a quantitatively similar number of updates of the estimated matrices A and b as in the online algorithms.

4.2. AOLSPI controlling the inverted pendulum

The second example system is the inverted pendulum with the parameters also taken from (Bušoniu et al., 2010). In order to balance the pendulum in the upright position, it is essential to use a continuous action-space representation; otherwise, undesired chattering around the unstable equilibrium will occur. Therefore, AOLSPI will be mainly compared to the relevant baseline algorithm OLSPI in this example.

The pendulum system consists of a DC-motor with a pole attached and the goal is to steer the pole into the upright position and balance it there. The dynamics are governed by

$$\ddot{\alpha} = \frac{1}{J}\left(mgl\sin(\alpha) - b\dot{\alpha} - \frac{K^2}{R}\dot{\alpha} + \frac{K}{R}u\right), \tag{36}$$

where α describes the current angle of the pole, $\dot{\alpha}$ the angular velocity, and $\ddot{\alpha}$ its angular acceleration. The values of the constants J, m, g, l, b, K, and R are set identically as in Bušoniu et al. (2010). The upright position is defined by $\alpha = 0$. For the simulation study, we employed a 4th order Runge–Kutta solver and a model discretization with sampling time $t_S = 0.005$ s. The variable $u \in \mathcal{A}_p$ denotes the input torque of the DC motor and is restricted to the continuous interval $\mathcal{A}_p = [-3\,\text{N m}, 3\,\text{N m}]$. The state $s = [\alpha, \dot{\alpha}]^\top$ of the inverted pendulum consists of the angle $\alpha \in [-\pi, \pi]$ and the angular velocity $\dot{\alpha}$, which is bounded by $|\dot{\alpha}| \le \alpha_{\max}$, $\alpha_{\max} = 15\pi\,\text{rad}\,\text{s}^{-1}$. In the following, the physical units are omitted for brevity and the quantities are given in SI unless stated differently. The state space of the system is given by $\mathcal{S}_p = [-\pi, \pi] \times [-15\pi, 15\pi]$. The reward function is chosen as $R_p(s, a) = -s^\top \text{diag}(5, 0.1)\, s - a^\top a$ and punishes angular deviations from the upright position, high angular velocities, and large control inputs.

In order to quantify the quality of a policy, we use the following metric: for a finite set of initial states $\mathcal{S}_0$, the average $\bar{G}_u$ of the total undiscounted sum of rewards obtained from all initial states of $\mathcal{S}_0$ when using the current policy for $N_{\text{test}} = 50$ time steps is calculated, i. e.,

$$\bar{G}_u = \frac{1}{|\mathcal{S}_0|} \sum_{i=1}^{|\mathcal{S}_0|} \sum_{j=1}^{N_{\text{test}}} R_j. \tag{37}$$

Note that this score function does not discount the rewards. The reward obtained when the pendulum is already swung up and needs to be balanced in the upright position is considered equally important during evaluation as the actual bang–bang like swing-up. Consequently, the effect of a discrete action set is not hidden from the performance score as it could occur with a discounted reward. As the initial state set $\mathcal{S}_0 \subset \mathcal{S}_p$, we distribute 35 states over $\mathcal{S}_p$ as

$$\mathcal{S}_0 = \{-\pi, -\frac{\pi}{2}, 0, \frac{\pi}{2}, \pi\} \times \{-10\pi, -3\pi, -\pi, 0, \pi, 3\pi, 10\pi\}.$$

The parameters of each algorithm evaluated in the simulation study are given in Table 5. To assess the performance of the algorithms, we evaluate $N_{\text{eval}} = 90$ independent runs per algorithm. Each run consists of 300 trials of 0.75 s of interaction, i. e., the system is reset to a random

Table 5
Parameters used in the inverted pendulum study.

Parameter	OLSPI	OKLSPI	aOLSPI
Discount factor γ	0.99	0.99	0.99
Exploration factor ε	0.95	0.95	0.95
Number of BFs $\lvert\phi\rvert$	11×11	dynamic	dynamic
RBF variance Σ	diag(0.2, 50)	diag(0.2, 50, 0.1)	diag(0.2, 50)
Coherence threshold μ	–	0.5	0.5
Update interval K_θ	5	5	5
Degree Chebyshev M	2	–	2
Action space $\mathcal{A}$	$\mathcal{A}_\mathrm{p}$	$[-3, 0, 3]$	$\mathcal{A}_\mathrm{p}$

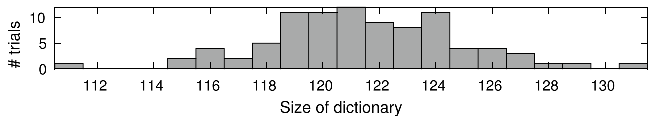


Fig. 4. Distribution of the size of the dictionary built by AOLSPI.

start state after $N_\mathrm{trial} = 150$ interactions. The exploration in time step t is again governed by (33), where $\varepsilon_\mathrm{min} = 0.05$ and $t_\mathrm{max} = 0.75\,\mathrm{s}$ is the duration of a single learning trial.

In order to compare the AOLSPI with its hand-tuned counterpart, let us consider the number and placement of the Gaussian BFs over the state space $\mathcal{S}_\mathrm{p}$. With the coherence threshold $\mu_0 = 0.5$, the AOLSPI algorithm creates dictionaries with $|\mathcal{D}| = 121.43$ elements on average; the distribution of the dictionary size over the 90 independent runs is depicted in Fig. 4. In order to compare the performance to that of OLSPI, we henceforth set the number of BFs to $N_\phi = 121$ and cover the state space with a regular grid. The resulting placement of the BFs is shown in Fig. 5. It can be observed that the automated kernel function selection by AOLSPI results in a less evenly distributed grid. However, the distance between each of the BFs is approximately similar when selected according to the coherence-based update rule (18). We also report our findings with the Babel criterion, cf. Remark 1. This sparsification rule is less suitable for online RL. Intuitively, this is because the BFs are not well spread over the state space. As can be seen in Fig. 5, rather many BFs are instead created along a particular trajectory until the threshold is reached; none can be added afterwards. Hence, the generalization capability of the value function Q suffers severely. This effect will not occur if *(i)* the data is supplied in random order to the learning algorithm or *(ii)* a suitable forgetting factor is included in the dictionary handling. In the design of OKLSPI and AOLSPI, neither is the case.

Next, the performance of the AOLSPI algorithm is investigated. Fig. 6 shows the mean score of the 90 independent runs for both the well-tuned OLSPI and the AOLSPI algorithms. On the one hand, with OLSPI it occurs easily that the performance is far worse than depicted; it is not obvious how to select the BF grid parameters appropriately beforehand. On the other hand, note that the placement as shown in Fig. 5 and overall necessary number of BF is obtained automatically by AOLSPI. Performance does not suffer from this online BF selection mechanism if the information spreading mechanism from Section 3.3 is employed. It is also confirmed that the initialization of new matrix/vector entries without extrapolation from previous iterations requires a much higher number of trials until convergence; in our simulation, AOLSPI without extrapolation does not even reach the same performance level within the given 300 trials.

The simulation results shown in Fig. 6 further underline the benefit of using a continuous action space representation for the pendulum problem. Note that the performance is measured according to (37), i. e., undesired chattering of the pendulum around the unstable equilibrium is notably penalized. Hence, although the OKLSPI algorithm fully uses the kernel trick, it fails to reach a similar level of performance as the other algorithms which employ the continuous action space approximation based on Chebyshev polynomials.

We now examine the influence of the extrapolation from Section 3.3 closer w. r. t. the performance of AOLSPI. In order to assess the influence, we performed additional runs with AOLSPI and the trust radius varying between only a little ($p_\mathrm{e} = 0.1$), a medium amount ($p_\mathrm{e} = 0.5$), and nearly full ($p_\mathrm{e} = 0.9$) extrapolation. The results are shown in Fig. 7. All existing BFs may be used to build $\tilde{D}$ in this particular simulation study. This is expected due to the Gaussian kernel (34) and the spread according to Table 5, which yields low correlations quickly for distant BFs. If, depending on the parameters, the information is not well spread during the dictionary update, it may nonetheless be useful to set $p_\mathrm{e} < 1$.

4.3. *Additional discussion of the similarity-based extrapolation*

With the simulation results reported above, the utility of the proposed TD information update rule is already evident. We nonetheless discuss in closer detail how (31) and (32) predict useful values for the initialization after the dictionary expansion, hence allowing for more efficient TD updates. Unfortunately, a quantitative evaluation of the extrapolation is not feasible because no ground truth is available for yet incompleted dictionaries. Instead, we exemplarily examine the estimation of $A_{t,\mathrm{new}\,(i,i)}$ and $b_{t+1,\mathrm{new}\,(i)}$ in an a posteriori analysis. To this end, we consider one of the matrices explicitly. Let us take A_{150} and b_{150} at the end ($t = 150$) of run 1, trial 1. Given $M = 2$ and $N_\mathrm{S} = |\mathcal{D}| = 121$ at the end of this trial, we have $A_{150} \in \mathbb{R}^{363\times363}$ and $b_{150} \in \mathbb{R}^{363}$. The (diagonal) values of A_{150} and b_{150} are now one after another set to zero and estimated according to (31) and (32), based on the remaining (diagonal) values of A_{150} and b_{150}. The result is illustrated in Fig. 8. It can be seen that the similarity weighting interpolation approach can reflect the trend of the elements of A and b, although the peaks may be missed. As expected, the estimates are rather conservative because (31) and (32) essentially compute locally weighted means, i. e., the relevant neighborhood is determined by the variance of the BFs functions. Hence, in order to capture either highly varying or very smooth relations in A and b, one would be forced to tune the variances. At this point, one would not reduce the burden of parameter tuning by means of this approach. However, as shown by Fig. 7, it is sufficient to add a rough prediction to improve the convergence speed. In summary, the diagonal similarity-weighting extrapolation (31) and (32) constitutes a simple yet efficient method to accelerate the online learning process in the face of dynamic dictionary growth.

5. Summary and future work

We investigate the well-known least-squares policy iteration algorithms KLSPI and OLSPI in view of their applicability to intelligent real-time automation, e. g., robotic control problems. The KLSPI algorithm is reformulated for incremental data collection, yielding the proposed OKLSPI for online usage. To this end, we adopt an efficient sparsification scheme from kernel adaptive filtering and derive a recursive dictionary expansion scheme with corresponding parameter update rule. The OLSPI can be endowed with an automatic basis function selection method by a similar course of action, effectively reducing the amount of required hand-tuning. The resulting AOLSPI algorithm is applicable to continuous state–action domains as well. A similarity-based TD information extrapolation scheme recovers the learning performance of the basic algorithms and we show that the convergence properties remain unaffected by our modifications. The utility of the novel algorithms is finally demonstrated by means of an illustrative simulation study.

The proposed algorithms constitute within the value function based approaches a further step towards the important goal of powerful online learning robot control. While the novel AOLSPI algorithm allows for continuous action space representations, this is not yet the case for OKLSPI, leaving room for future work. Moreover, automating the selection of the kernel hyper-parameters remains an important yet in general challenging research question.

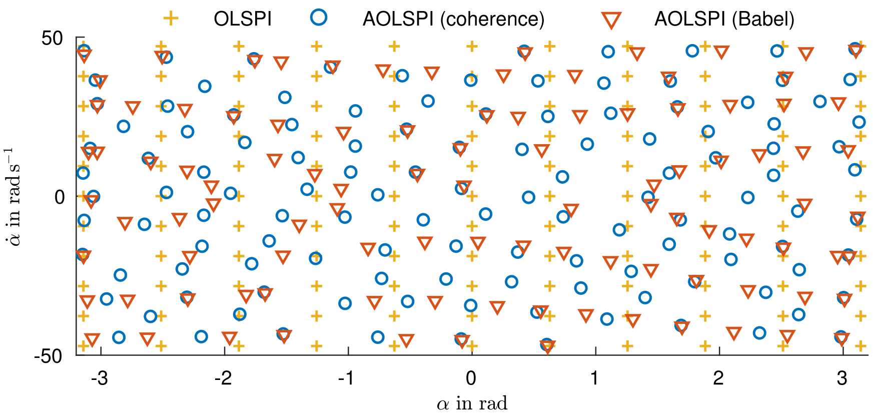


Fig. 5. Placement of the BFs over the state space $\mathcal{S}_\mathrm{p}$. The grid had to be set manually for OLSPI (yellow crosses), whereas the AOLSPI VFA bases were obtained automatically. Note that the typical inverted pendulum traces become visible using the Babel criterion (red triangles), whereas coherence sparsification (blue circles) leads to a good approximation throughout the state space.

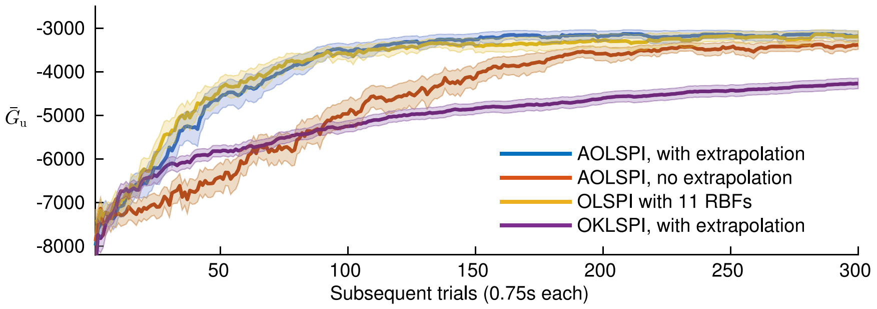


Fig. 6. Performance comparison of OLSPI and AOLSPI. The figure depicts the mean score according to (37) over the 90 runs (thick lines) and the corresponding 95% confidence intervals (shaded areas). The TD update information extrapolation after insertion of a new dictionary element is according to Section 3.3.

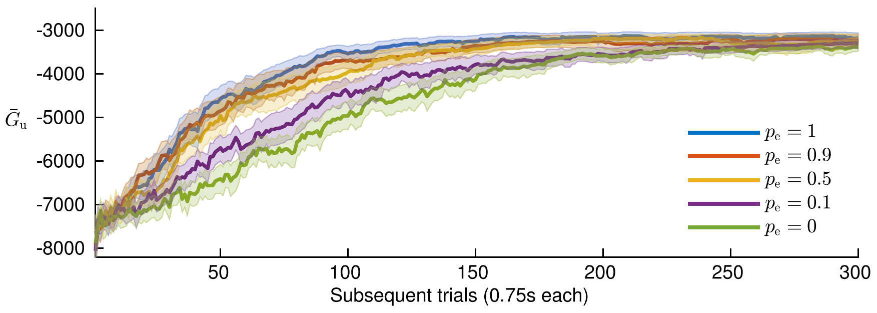


Fig. 7. Effect of the trust radius on AOLSPI learning performance. The graph depicts the quality of the policy in the subsequent trials computed according to (37). A clear improvement in convergence is apparent for approximately $p_\mathrm{e} \geq 0.5$, i.e., the 50% most similar features are used for information extrapolation according to (31)–(32).

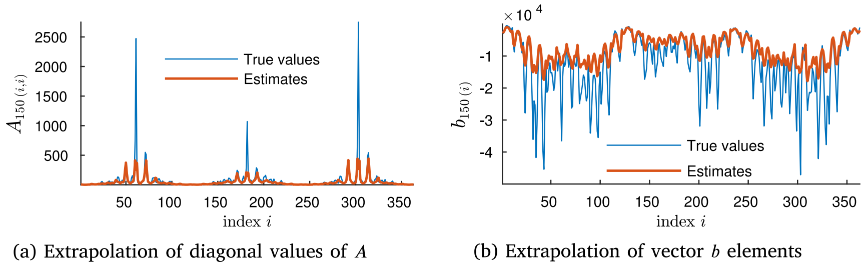


Fig. 8. As no ground truth is available to reflect the online situation, this graph shows an a posteriori comparison of estimated diagonal entries of A_{150} and estimated entries of b_{150} w.r.t. their true values. Although this comparison cannot accurately reflect the situation during the online algorithmic execution, it is apparent that the corresponding values will be predicted correctly to a certain extent.

Acknowledgments

This work was supported in part within the ERC Advanced Grant SHRINE Agreement No. 267877 and in part by the Technische Universität München–Institute for Advanced Study (www.tum-ias.de), funded by the German Excellence Initiative.

References

Alibekov, E., Kubalík, J., Babuška, R., 2018. Policy derivation methods for critic-only reinforcement learning in continuous spaces. Eng. Appl. Art. Intell. 69, 178–187.

Anderlini, E., Forehand, D.I.M., Bannon, E., Abusara, M., 2017. Control of a realistic wave energy converter model using least-squares policy iteration. IEEE Trans. Sustain. Energy 8 (4), 1618–1628.

Bellman, R., 1957. Dynamic Programming. Princeton University Press, NJ USA.

Bertsekas, D.P., 1995. Dynamic programming and optimal control, vol. 2, 1. Athena Scientific Belmont, MA.

Buşoniu, L., Ernst, D., Schutter, B.D., Babuška, R., 2010. Online least-squares policy iteration for reinforcement learning control, In: Proc. Amer. Control Conf., ACC, 2010, pp. 486–491.

Buşoniu, L., Lazaric, A., Ghavamzadeh, M., Munos, R., Babuška, R., De Schutter, B., 2012. Least-squares methods for policy iteration. In: Wiering, M., van Otterlo, M. (Eds.), Reinforcement Learning: State-of-the-Art. In: Adaptation, Learning, and Optimization, vol. 12, Springer, Heidelberg, Germany, pp. 75–109.

Bušoniu, L., Babuška, R., De Schutter, B., Ernst, D., 2010. Reinforcement Learning and Dynamic Programming Using Function Approximators, vol. 39. CRC press.

Cui, Y., Matsubara, T., Sugimoto, K., 2017. Kernel dynamic policy programming: Applicable reinforcement learning to robot systems with high dimensional states. Neural Netw. 94, 13–23.

Deisenroth, M.P., Fox, D., Rasmussen, C.E., 2015. Gaussian processes for data-efficient learning in robotics and control. IEEE Trans. Pattern Anal. Mach. Intell. 37 (2), 408–423.

Deisenroth, M.P., Neumann, G., Peters, J., 2013. A survey on policy search for robotics. Found. Trends Robot. 2 (1–2), 1–142.

Farahmand, A., Ghavamzadeh, M., Szepesvári, C., Mannor, S., 2016. Regularized policy iteration with nonparametric function spaces. J. Mach. Learn. Res. 17 (139), 1–66.

Friedrich, S.R., Buss, M., 2017. A robust stability approach to robot reinforcement learning based on a parameterization of stabilizing controllers, In: IEEE Int. Conf. Robot. Autom., ICRA, 2017, pp. 3365–3372.

Geramifard, A., Walsh, T.J., Tellex, S., Chowdhary, G., Roy, N., How, J.P., 2013. A tutorial on linear function approximators for dynamic programming and reinforcement learning. Found. Trends Mach. Learn. 6 (4), 375–451.

Honeine, P., 2015. Approximation errors of online sparsification criteria. IEEE Trans. Signal Process. 63 (17), 4700–4709.

Hourfar, F., Bidgoly, H.J., Moshiri, B., Salahshoor, K., Elkamel, A., 2019. A reinforcement learning approach for waterflooding optimization in petroleum reservoirs. Eng. Appl. Art. Intell. 77, 98–116.

Huang, Z., Xu, X., He, H., Tan, J., Sun, Z., 2017. Parameterized batch reinforcement learning for longitudinal control of autonomous land vehicles. IEEE Trans., Syst. Man, Cybern., Syst. PP (99), 1–12.

Jakab, H.S., Csató, L., 2015. Sparse approximations to value functions in reinforcement learning. In: Koprinkova-Hristova, P., Mladenov, V., Kasabov, N.K. (Eds.), Artificial Neural Networks. Springer, Cham, pp. 295–314.

Jung, T., Polani, D., 2007. Kernelizing LSPE(λ), In: IEEE Int. Symp. ADPRL, 2007, pp. 338–345.

Kober, J., Bagnell, D., Peters, J., 2013. Reinforcement learning in robotics: A survey. Int. J. Robot. Res. 32 (11), 1238–1274.

Lagoudakis, M.G., Parr, R., 2003. Least-squares policy iteration. J. Mach. Learn. Res. 4, 1107–1149.

Liu, W., Principe, J.C., Haykin, S., 2011. Kernel Adaptive Filtering: A Comprehensive Introduction, vol. 57. John Wiley & Sons.

Ma, J., Powell, W.B., 2009. A convergent recursive least squares approximate policy iteration algorithm for multi-dimensional Markov decision process with continuous state and action spaces, In: IEEE Int. Symp. ADPRL, 2009, 66–73.

Ma, J., Powell, W.B., 2010. Convergence analysis of kernel-based on-policy approximate policy iteration algorithms for Markov decision processes with continuous, multidimensional states and actions. In: Dept. Oper. Res. Financial Eng. Princeton Univ, pp. 1–40.

Mori, T., Howard, M., Vijayakumar, S., 2011. Model-free apprenticeship learning for transfer of human impedance behaviour, In: 11th IEEE-RAS Int. Conf. Humanoid Robots, 2011, pp. 239–246.

Ormoneit, D., Sen, Ś., 2002. Kernel-based reinforcement learning. Mach. Learn. 49 (2), 161–178.

Palunko, I., Donner, P., Buss, M., Hirche, S., 2014. Cooperative suspended object manipulation using reinforcement learning and energy-based control. In: 2014 IEEE/RSJ Int. Conf. Intell. Robot Syst. IROS, pp. 885–891.

Palunko, I., Faust, A., Cruz, P., Tapia, L., Fierro, R., 2013. A reinforcement learning approach towards autonomous suspended load manipulation using aerial robots, In: IEEE Int. Conf. Robot. Autom., ICRA, 2013, pp. 4896–4901.

Pan, Z., Chen, H., You, X., 2012. Support vector machine with orthogonal legendre kernel, In: Int. Conf. Wavelet Anal. Pattern Recogn., 2012, 125–130.

Pane, Y.P., Nageshrao, S.P., Kober, J., Babuška, R., 2019. Reinforcement learning based compensation methods for robot manipulators. Eng. Appl. Artif. Intell. 78, 236–247.

Polydoros, A.S., Nalpantidis, L., 2017. Survey of model-based reinforcement learning: Applications on robotics. J. Intell. Robot. Syst. 86 (2), 153–173.

Powell, W.B., Ma, J., 2011. A review of stochastic algorithms with continuous value function approximation and some new approximate policy iteration algorithms for multidimensional continuous applications. J. Control Theory Appl. 9 (3), 336–352.

Puterman, M.L., 1994. Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons.

Rasmussen, C.E., Williams, C.K.I., 2006. Gaussian processes for machine learning. In: Adaptive Computation and Machine Learning. MIT Press.

Richard, C., Bermudez, J.C.M., Honeine, P., 2009. Online prediction of time series data with kernels. IEEE Trans. Signal Process. 57 (3), 1058–1067.

Schölkopf, B., Smola, A.J., 2002. Learning with Kernels: Support vector machines, regularization, optimization, and beyond. In: Adaptive Computation and Machine Learning Series. MIT Press.

Stulp, F., Sigaud, O., 2013. Robot skill learning: From reinforcement learning to evolution strategies. Paladyn J. Behav. Robot. 4 (1), 49–61.

Sutton, R.S., Barto, A.G., 1998. Reinforcement Learning: An Introduction. MIT press, Cambridge.

Sutton, R.S., Barto, A.G., Williams, R.J., 1992. Reinforcement learning is direct adaptive optimal control. IEEE Control Syst. 12 (2), 19–22.

Taylor, G., Parr, R., 2009. Kernelized value function approximation for reinforcement learning. In: Proc. 26th Int. Conf. Mach. Learn., ICML. ACM, pp. 1017–1024.

Tolić, D., Palunko, I., 2017. Learning suboptimal broadcasting intervals in multi-agent systems. IFAC-PapersOnLine 50 (1), 4144–4149, 20th IFAC World Congress.

Tziortziotis, K., Vlachos, K., Blekas, K., 2016. Reinforcement learning-based motion planning of a triangular floating platform under environmental disturbances, In: 24th Medit. Conf. Control Autom., MED, 2016, pp. 1014–1019.

Vankadari, M.B., Das, K., Shinde, C., Kumar, S., 2018. A reinforcement learning approach for autonomous control and landing of a quadrotor, In: Int. Conf. Unmanned Aircraft Syst., ICUAS, pp. 676–683.

Vinogradska, J., Bischoff, B., Peters, J., 2018. Approximate value iteration based on numerical quadrature. IEEE Robot. Autom. Lett. 3 (2), 1330–1337.

Vrabie, D., Vamvoudakis, K.G., Lewis, F.L., 2012. Optimal adaptive control and differential games by reinforcement learning principles. In: Control, Robotics & Sensors. IET.

Wang, J., Xu, X., Liu, D., Sun, Z., Chen, Q., 2014. Self-learning cruise control using kernel-based least squares policy iteration. IEEE Trans. Control Syst. Technol. 22 (3), 1078–1087.

Xu, X., Hou, Z., Lian, C., He, H., 2013. Online learning control using adaptive critic designs with sparse kernel machines. IEEE Trans. Neural Netw. Learn. Syst. 24 (5), 762–775.

Xu, X., Hu, D., Lu, X., 2007. Kernel-based least squares policy iteration for reinforcement learning. IEEE Trans. Neural Netw. 18 (4), 973–992.

Xu, X., Lian, C., Wang, J., He, H.-G., Hu, D., 2016. Actor–critic reinforcement learning for autonomous control of unmanned ground vehicles. Sci. Robot. 42.

Xu, X., Lian, C., Zuo, L., He, H., 2014. Kernel-based approximate dynamic programming for real-time online learning control: An experimental study. IEEE Trans. Control Syst. Technol. 22 (1), 146–156.

Yahyaa, S., Manderick, B., 2014. Knowledge gradient for online reinforcement learning. In: Duval, B., van den Herik, J., Loiseau, S., Filipe, J. (Eds.), Agents and Artificial Intelligence. In: ICAART 2014 LNCS, vol. 8946, Springer, Cham, pp. 103–118.