Engineering Applications of Artificial Intelligence 107 (2022) 104515

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

Adaptive polyhedral meshing for approximate dynamic programming in control

Antonio Sala [a,*], Leopoldo Armesto [b]

[a] *Instituto Universitario de Automática e Informática Industrial (AI^2), Universitat Politècnica de València, Camino de Vera s/n, Valencia, 46022, Valencia, Spain*
[b] *Instituto de Diseño y Fabricación (IDF), Universitat Politècnica de València, Camino de Vera s/n, Valencia, 46022, Valencia, Spain*

ARTICLE INFO

Keywords:
Optimal control
Dynamic programming
Function approximation

ABSTRACT

This work proposes a new criterion for adaptive meshing in polyhedral partitions which interpolate a value function in Approximate Dynamic Programming (ADP) in optimal control problems. The criterion adds new points to a simplicial mesh, based on: a user-defined initial condition probability density function which determines 'influential' regions of the state space, uncertainty (variance) propagation, and temporal-difference error. A collection of lemmas justifies the algorithmic proposal. Comparative analysis with other options in literature highlights the advantages of our proposal. The developed methods are applied to simulation examples and an experimental robotic setup.

1. Introduction

Optimal control laws for discrete-time dynamic systems can be designed via various techniques: dynamic programming (DP), reinforcement learning (RL) and other techniques such as model predictive control (MPC). DP will be understood as a model-based approach with all transitions, rewards and states assumed explicitly available; RL will be addressed to solving similar problems in the absence of a model, via "experimentation" (actual or simulated) of state transitions and rewards (Maggipinto et al., 2020; Busoniu et al., 2018); MPC carries out on-line optimization (Altan and Hacıoğlu, 2020).

The goal of this work is restricted to the first of the frameworks, i.e., dynamic programming: model-based DP avoids on-line computational burden of MPC at the expense, of course, of incurring in such a burden in the controller design phase. If the state space is finite and each state has an associated value function parameter, it is well known that standard policy iteration (PI) and value iteration (VI) techniques do converge to the optimal solution under mild assumptions (Bertsekas, 2017). In the case of continuous state spaces, approximate dynamic programming (ADP) is needed in all but a few simple cases, but in such a case, convergence guarantees may be lost (Busoniu et al., 2010).

One popular setting in which ADP convergence is guaranteed is the so-called fuzzy, interpolated or gridded approach (Davies, 1996; Grüne, 1997; Busoniu et al., 2010). Basically, the continuous state space is discretized and a finite-state MDP problem is solved on the discretized approximation. This is, too, related to the aggregation/disaggregation approaches to ADP (Bertsekas, 2018). Other ADP approaches might not have the required contractiveness requirements for PI/VI convergence and alternative solutions may be needed, such as linear programming (De Farias and Van Roy, 2003; Díaz et al., 2020) or numerical minimization of the Bellman error (fitted policy iteration), see Antos et al. (2006), Timmer and Riedmiller (2007) and Díaz et al. (2020); however, the computation time requirements the latter options are significantly larger compared to the aforementioned interpolated ones.

Once an ADP solution is found, there are approaches in literature modifying the function approximation structure, trying to find a more accurate approximation in later repetitions of the PI/VI step, see Grüne (1997), Munos and Moore (2002), Keller et al. (2006) and Whiteson (2006). The most naive option would be uniform state–space discretization with finer granularity in a simplicial interpolation mesh, for instance; increasing the granularity of the action-space might also be considered (Buşoniu et al., 2018; Farquhar et al., 2020). Of course, uniform gridding renders impractical as the density of sample points increases, this issue being exacerbated with the dimension of the state space. However, non-uniform simplicial (or, in general, polyhedral) meshes are an interesting option for ADP and, in fact, this manuscript will concentrate on their refinement as its main objective.

Notwithstanding, for completeness of the literature review, there are other possible approaches in order to adapt the ADP function approximators: Keller et al. (2006) discusses a regression-based approach in with basis functions (features) are automatically added, evolving neural network architectures are proposed in Whiteson (2006), tile-coding structures appear in Whiteson et al. (2007), and multi-scale kd-trees in Gomez Plaza et al. (2017). Specialized mesh-refinement algorithms for particular optimal control problem cases are also a

* Corresponding author.
E-mail address: asala@isa.upv.es (A. Sala).

https://doi.org/10.1016/j.engappai.2021.104515
Received 30 July 2021; Received in revised form 22 September 2021; Accepted 14 October 2021
Available online 29 October 2021

possible option: for instance, in Yershov and Frazzoli (2015), specializations to the shortest-path problem in robotic planning are proposed, nevertheless, they do not apply to a general DP setting. Last, in a continuous-time setting, the Hamilton–Jacobi-Bellman equation at the core of DP is actually a first-order partial differential equation that can be solved by, say, collocation methods: the works (Liu et al., 2015, 2017) exploit such an option; nevertheless, the result is obtained in the time domain (meshing is done in time intervals, interpolation of control actions in between the intervals is done with, say, Legendre polynomials of varying degree) so their developments do not obtain value functions $V(x)$, being thus more related to open-loop control computations. The mentioned neural, evolving, problem-specific or PDE-collocation approaches will not be considered in the scope of this work, due to their computational costs or lack of applicability to generic closed-loop optimal control.

Hence, motivated on the sensible computational cost and intuitive interpretation of the interpolation setups, we will be focusing in simplicial mesh refinement criteria for value function and state-feedback control computation. Non-uniform polyhedral meshes for interpolation will be the chosen tool in this work, because they preserve the favorable properties for ADP convergence and, at least in theory, they can account for the different mesh density needs in different regions of the state space, depending on the variations of the smoothness of the approximated value function.

In order to generate non-uniform regressor arrangements, there exist naive approaches such as logarithmically-spaced options (Busoniu et al., 2010); other possible options are based only in a function approximation paradigm (Cohen et al., 2012), disregarding the fact that the approximated function comes from a DP setup. However, in a dynamic programming setting, introducing specific measures related to optimal control performance proves advantageous (Grüne, 1997). The cited work, and the refinements in Grüne and Semmler (2004), Armesto and Sala (2022), propose a splitting criterion based on the Bellman temporal-difference error (TDE). In Munos and Moore (2002) and Cervellera and Macciò (2017) the importance of adapting the meshing (or sample points, in the second case) to the distribution of state trajectories is acknowledged, identifying influential regions of the state space (Gottesman et al., 2020). The work Munos and Moore (2002) is a very comprehensive one in which a plethora of options for adaptive meshing in DP problems are compared in different test benches. Comparative discussion of our proposal and those in the above-cited works will be carried out in later examples and discussion sections in this manuscript.

The main objective of this paper is proposing a splitting criterion for polyhedral interpolation meshes used in ADP, improving upon the above-cited literature proposals on this topic. The algorithmic choices are justified by a collection of lemmas on uncertainty/influence propagation in dynamic programming problems. In particular, we propose a mixture random variable framework giving a probabilistic interpretation to the square of the value function approximation error (decomposed as bias plus variance terms); also, we propose adding spatial-dependent weights (influence) to adapt computation accuracy to a given probability distribution of initial conditions. The validity of these ideas will be discussed, comparing them with prior literature, and illustrated in some examples, including a robotic experiment on path-planning in configuration space.

The structure of this work is as follows: next section discusses preliminaries, notation and problem statement; Section 3 discusses the main result and lemmas supporting it; Section 4 provides some examples; comparative analysis and discussion appear in Section 5, and a conclusions section closes the paper.

2. Preliminaries and notation

Let us consider a discrete-time dynamic system:

$$x^+ = f(x,u) \tag{1}$$

where $x \in \mathbb{X}$ is the state vector, $u \in \mathbb{U}$ is the input (manipulated variable, $\mathbb{U}$ being the set of valid control actions) and $x^+ \in \mathbb{X}\cup\mathbb{T}$ denotes the next (successor) state. The state space $\mathbb{X} \subset \mathbb{R}^n$ will be assumed to be a compact polyhedron, where $\mathbb{T} \subseteq \mathbb{R}^n \sim \mathbb{X}$ is denoted as *terminal set*.

A stationary policy $u = \pi(x)$ (i.e., a time-invariant state-feedback controller) is a function $\mathbb{X} \to \mathbb{U}$ that represents a control law such that the closed-loop system achieves the time-invariant dynamics $x^+ = f(x,\pi(x))$. The cost $V^\pi : \mathbb{X}\cup\mathbb{T} \mapsto \mathbb{R}$, associated to a policy $\pi(x)$ starting from an initial state x_0 will be defined as:

$$V^\pi(x_0) := V_{\mathbb{T}}(x_{t_T(x_0)}) + \sum_{k=0}^{t_T(x_0)-1} \gamma^k L(x_k,\pi(x_k)), \tag{2}$$

being $x_k = f(x_{k-1},\pi(x_{k-1}))$ the state at time instant $k \geq 1$, and $t_T(x_0)$ is the time instant in which $x_{t_T(x_0)}$ reaches $\mathbb{T}$; if no such time instant exists, then $t_T(x_0) = \infty$ and $V_{\mathbb{T}}(\cdot)$ will be irrelevant to the cost computation. Note that if $x_0 \in \mathbb{T}$ then $t_T(x_0) = 0$ and only the terminal cost is evaluated.

In the above expression $L : \mathbb{X}\times\mathbb{U} \to \mathbb{R}$ is a scalar function, bounded in $\mathbb{X}\times\mathbb{U}$, also known as "immediate" or "stage" cost and $0 < \gamma < 1$ is a discount factor, and $V_{\mathbb{T}} : \mathbb{T} \to \mathbb{R}$ is denoted as "terminal cost".

The goal of dynamic programming is finding an optimal policy $\pi^*(x)$ minimizing $V^\pi(x)$. The cost of an arbitrary policy must satisfy Bellman equation (Bertsekas, 2017):

$$V^\pi(x) = L(x,\pi(x)) + \gamma V^\pi(f(x,\pi(x))) \quad \forall x \in \mathbb{X}. \tag{3}$$

The value function of the optimal policy, denoted as $V^*(x)$, verifies:

$$V^*(x) = TV^*(x), \tag{4}$$

where the Bellman operator T acting on a function $V : \mathbb{X}\cup\mathbb{T} \mapsto \mathbb{R}$ is defined as the function $TV : \mathbb{X} \mapsto \mathbb{R}$ given by:

$$TV(x) := \min_{u\in\mathbb{U}} \left(L(x,u) + \gamma V(f(x,u))\right). \tag{5}$$

Now, from $V^*(x)$, the optimal policy can be obtained:

$$\pi^*(x) = \arg\min_{u\in\mathbb{U}} \left(L(x,u) + \gamma V^*(f(x,u))\right). \tag{6}$$

Exact dynamic programming. If both the state space $\mathbb{X}$ and action space $\mathbb{U}$ are finite, under mild assumptions, the operator T is a contraction and the value iteration algorithm $V_k(x) = TV_{k-1}(x)$ converges to a fixed point which is coincident with the optimal value function; other well-known algorithms, such as policy iteration and linear programming, can also be successfully used in the finite case (De Farias and Van Roy, 2003; Busoniu et al., 2010; Bertsekas, 2017).

2.1. Approximate dynamic programming

Now, let us consider a function approximator $V(x,\theta)$ where $\theta \in \mathbb{R}^m$ is a vector of adjustable parameters. Based on Bellman's Eq. (3), given a policy $\pi(x)$, if we define the Bellman residual (Lagoudakis and Parr, 2003), also denoted as temporal-difference error (TDE) (Maei et al., 2009), as:

$$\epsilon^\pi(x,\theta) := V^\pi(x,\theta) - L(x,\pi(x)) - \gamma V^\pi(f(x,\pi(x)),\theta). \tag{7}$$

Then, the approximate least-squares solution to the "policy evaluation" problem, given a policy $\pi(x)$, is given by:

$$\hat{\theta}^\pi := \arg\min_\theta \|\epsilon^\pi(x,\theta)\|^2, \tag{8}$$

where $\|\epsilon^\pi(x,\theta)\|^2 := \int_{\mathbb{X}} \epsilon^\pi(x,\theta)^2$. Obviously, the integral should be understood as a sum in discrete (finite) state spaces; also, in continuous state–spaces, to avoid numerical integration steps, the integral is often understood as the sum of $\epsilon^{\pi}(x_k, \theta)^2$ over a set of available fitting (training) points $\{x_1, \dots, x_N\}$, see Antos et al. (2007), for instance.

The motivation to the above optimization problem (8) lies in the fact that, if $\epsilon^{\pi}(x) = 0$ for all $x \in \mathbb{X}$, then (3) is fulfilled, so we have accurately computed the value function of the policy. Of course, checking the whole state space $\mathbb{X}$ can only be done in a finite case, but a low fitting error (8) with a large N should reasonably approach the ideal perfect-fit situation.

In order to find an approximate optimal value function (and its associated policy), we can redefine (7) as:

$$\epsilon(x, \theta) := V(x, \theta) - \min_{u \in \mathbb{U}} (L(x, u) + \gamma V(f(x, u), \theta)) \tag{9}$$

and try to obtain θ that minimizes $\|\epsilon(x, \theta)\|^2$.

In the sequel, from (6), for a given (maybe sub-optimal) value function $V(x, \theta)$, we will define the policy associated to $V(x, \theta)$ as:

$$\hat{\pi}(x, \theta) := \arg\min_{u \in \mathbb{U}} (L(x, u) + \gamma V(f(x, u), \theta)). \tag{10}$$

As mentioned in the introduction, in a generic case, the so-called (fitted) Policy Iteration (PI) and (fitted) Value iteration (VI), which are popular algorithms to solve the ADP problem (Díaz et al., 2020; Munos and Szepesvári, 2008), do not converge with arbitrary parametrizations. Only in very particular scenarios, where the composition of the Bellman operator T in (5), plus "projection" (8) to the space of functions that can be represented by the chosen parametrization, is contractive, we can guarantee convergence to a given "fixed point" (Busoniu et al., 2010; Bertsekas, 2017). Contractiveness has been only proved in some particular cases, such as using a Look-Up-Table (LUT) with as many regressors as available data, see Busoniu et al. (2010) for details. Apart from regular discretizations of the state space (which easily fall into the *curse of dimensionality* for high dimensional systems), other approximation options such as fuzzy interpolation, tile coding (Whiteson et al., 2007; Sherstov and Stone, 2005), simplicial/polyhedral partitions (Grüne, 1997; Munos and Moore, 2002), etc. also fulfill the contractiveness requirement. In this work, we will restrict to refinement of simplicial/polyhedral interpolation structures, to be described next. Some proposals in this manuscript might be amenable to being used, too, in other of the above function approximation options, but this has been left for future work.

2.2. Polyhedral partitions for function approximation

Polyhedral partitions. We will assume that we have a finite set of known 'vertex' points in a source dataset $\mathbb{D} = \{x_1, \dots, x_N\}$. Let us consider the state–space $\mathbb{X}$ partitioned in a collection of polyhedral "cells" H_m, so $\mathbb{X} = \bigcup_m H_m$, with $int(H_i) \cap int(H_j) = \emptyset\ \forall i, j$, disjoint interiors, with all vertices of H_m belonging to $\mathbb{D}$ and, vice-versa, each x_i in $\mathbb{D}$ belonging to at least one H_m, see Fig. 1.

Barycentric coordinates. We will assume that there exist interpolation functions $h_k(x) : \mathbb{X} \mapsto \mathbb{R}^+$, for $k = 1, \dots, N$, such that $x \equiv \sum_{k=1}^N h_k(x) \cdot x_k$, with $0 \le h_k(x) \le 1$, $\sum_{k=1}^N h_k(x) = 1$ for every $x \in \mathbb{X}$. The assumption is trivially true for the above-described polyhedral partition, due to the convexity of each of the cells, details left to the reader. A polyhedral partition is said to be *simplicial* if the number of nonzero $h_k(x)$ is at most $n + 1$ for all $x \in \mathbb{X}$, such as the one in Fig. 1, with $n = 2$. In intelligent control literature, $h_k(\cdot)$ are sometimes denoted as 'membership' or 'activation' functions.

Local patch. Given a point $x_k \in \mathbb{D}$, its local patch $\mathcal{L}_k$ will be defined as the support of $h_k(x)$, i.e., $\mathcal{L}_k := \{x | h_k(x) > 0\}$.

Interpolation. If the values of a function at the partition vertex points $V_k := V(x_k)$ are known for all $x_k \in \mathbb{D}$, and $h_k(x_k) = 1$, the function at intermediate points will be assumed to be interpolated as

$$\hat{V}(x, \theta) := \sum_{k=1}^{N} h_k(x) \cdot V_k, \tag{11}$$

where the parameter vector is $\theta := \{V_1, \dots, V_N\}$.

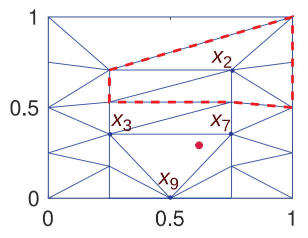


Fig. 1. Example 25-point simplicial mesh: the red dot can be expressed as $h_7 x_7 + h_3 x_3 + h_9 x_9$, with $h_7 = 0.69$, $h_3 = 0.18$, $h_9 = 0.13$, the rest of barycentric coordinates being zero; the dashed red polygon delimits the local patch of x_2. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Interpolated TDE. Under this assumption, the ADP temporal-difference error (9) will be written as:

$$\epsilon(x, \theta) = \sum_{k=1}^{N} h_k(x) V_k - \min_{u \in \mathbb{U}} \left(L(x, u) + \gamma \sum_{k=1}^{N} h_k(f(x, u)) V_k \right). \tag{12}$$

In the sequel, we will omit the parameterized notation with argument θ, and we will denote the left-hand side of (11) and (12) as $\hat{V}(x)$ and $\epsilon(x)$, respectively, if θ is clear from the context.

2.3. Adaptive meshing

Once a successfully converged ADP value function $\hat{V}(x)$ has been obtained, accuracy information may be extracted from it in some ways, to guide future modifications to the partition/regressor setup. Basically, in adaptive polyhedral partition setups, some "cells" are split based on a given criterion, which may be classified, according to Munos and Moore (2002), for instance, as:

- *local*: in which a Bellman error (Grüne, 1997; Whiteson et al., 2007), slope or curvature measure (Munos and Moore, 2002), or policy disagreement between different vertices of a cell (Munos and Moore, 2002) can be obtained by looking at a cell's vertices or a handful of close points,
- *global*: where the accumulation of error or uncertainty along trajectories is taken into account so discounted-sum formulae for such criteria (purposely resembling (2), the actual cost index) can be proposed (Munos and Moore, 2002).

2.4. Problem statement

Concentrating on local-only adaptive meshing criteria above might be misled by discontinuities or sharp value function changes in small regions of the state space that might be not so relevant for a given application, this is why global considerations are of interest, motivating this work.

In this paper we will propose a criterion for addition of points in adaptive meshing taking into account the influence of a given candidate cell splitting on the propagation of a quadratic error measure. The said propagation will be computed taking into account a given set (or probability distribution) of initial conditions where good accuracy in value function estimation is pursued. Comparative analysis with other options in literature will be also carried out.

This work will only discuss addition of points to the mesh; of course, adaptive meshing methods may be more efficient if they include a clean-up step where deletion of a point (and a possible re-triangulation) is decided if it induces a low fitting error. As these error ideas are related to plain function approximation, not particularly dependent on features of dynamic programming problems, mesh coarsening is not considered in the scope of this work, for brevity.

3. Main result

3.1. Problem discretization

Let us assume that we have a polyhedral partition of the continuous-time state space, as discussed in Section 2.2. Now, let us also assume that we have a finite set of points in a source dataset $\mathbb{D} = \{x_1, \dots, x_N\}$, a finite set of control actions $\mathbb{U} = \{u_1, \dots, u_M\}$ and the associated successors for all state and control combinations $x_{ij}^+ = f(x_i, u_j)$. Of course, each of these successors who lies in $\mathbb{X}$ can be expressed in barycentric coordinates $x_{ij}^+ = \sum_{k=1}^N h_k(x_{ij}^+) x_k$ with $0 \le h_k(x_{ij}^+) \le 1$ and $\sum_{k=1}^N h_k(x_{ij}^+) = 1$. For convenience, notation $h_{k,ij} := h_k(x_{ij}^+)$ will be used in the sequel, i.e., $x_{ij}^+ = \sum_{k=1}^N h_{k,ij} x_k$.

Now, consider the approximate Bellman equation for the optimal policy at source points

$$V(x_i) = \min_j \left(L(x_i, u_j) + \gamma V(x_{ij}^+) \right) \qquad x_i \in \mathbb{D}, \tag{13}$$

which, using the simplicial approximator (11), if $x_{ij}^+ \in \mathbb{X}$, renders equal to:

$$V(x_i) = \min_j \left(L(x_i, u_j) + \gamma \sum_{k=1}^N h_{k,ij} V(x_k) \right). \tag{14}$$

When $x_{ij}^+ \notin \mathbb{X}$, i.e., $x_{ij}^+ \in \mathbb{T}$, successors in the terminal set $\mathbb{T}$ need separate handling, of course, using $V_{\mathbb{T}}(x_{ij}^+)$ in (14) instead of the barycentric coordinates in such a case. Abusing the notation, this will be implicitly assumed in the sequel, instead of complicating notation in (14) and similar expressions with terminal vs. non-terminal cases, leaving such details to the reader.

If we intentionally switch to considering $h_{k,ij}$ as probabilities (instead of deterministic barycentric coordinates), we can reinterpret the above "deterministic" Bellman equation, now considering it to be the Bellman equation of a discrete stochastic Markov Decision Process (MDP) $x_{ij}^+ = z(x_i, u_j)$ being $z(x_i, u_j)$ a random variable that takes value $x_k \in \mathbb{D}$ with probability $h_{k,ij}$. Indeed, such Bellman equation is

$$V(x_i) = \min_j E\left[L(x_i, u_j) + \gamma V(x_{ij}^+) \right], \tag{15}$$

where $E[\cdot]$ denotes mathematical expectation and, indeed, it actually amounts to (14) once the expectation is replaced by the summation that computes it. This reinterpretation from barycentric coordinates to probabilities is what in DP literature is named as (approximate) *discretization*.

The "exact" value iteration on the above discrete MDP converges and, of course, it is numerically equivalent to the "approximate" dynamic programming solution of VI over a continuous domain $\mathbb{X}$ obtained with a simplicial function approximator (Busoniu et al., 2010; Bertsekas, 2018).

Continuous initial state (partial discretization). In later developments, we will "discretize" our continuous MDP after the first step, i.e., in order to approximate properties of the original continuous-state optimal control problem. Indeed, we will compute some statistical properties of an MDP such that the initial state is in $\mathbb{X}$, but its successor will be assumed to be a point in the finite source set $\mathbb{D}$, by reinterpreting barycentric coordinates of $x^+ = f(x,u) = \sum_{k=1}^N h_k(f(x,u)) \cdot x_k$ as probabilities, $x^+ = z(x,u)$ being z a random variable that takes value x_k with probability $h_k(f(x,u))$, in an analogous way as done with points in $\mathbb{D}$ earlier on, but keeping f to be a function of continuous-valued arguments.

3.2. Bellman temporal-difference error and approximation accuracy

Lemma 1. *Given a dataset $\mathbb{D}$, consider the discretized solution of (14) over it and its interpolation $\hat V(x)$ for arbitrary points in $\mathbb{X}$ given by (11). Denote as $\hat\pi(x)$ the approximation to the optimal policy associated to $\hat V(x)$ obtained from (10). Also, denote as $V^{\hat\pi}(x)$ the true value function of $\hat\pi(x)$. Then, the approximation error $\beta(x) := \hat V(x) - V^{\hat\pi}(x)$ between the true value function of said policy and the value-iteration approximation $\hat V(x)$ verifies:*

$$\beta(x) = \epsilon(x) + \gamma \beta(f(x, \hat\pi(x))), \tag{16}$$

where $\epsilon(x)$ is given in (12).

Proof. The true value function of $\hat\pi(x)$ fulfills the Bellman equation $V^{\hat\pi}(x) = L(x, \hat\pi(x)) + \gamma V^{\hat\pi}(x^+)$, with $x^+ = f(x, \hat\pi(x))$. Thus, the approximation error fulfills:

$$V^{\hat\pi}(x) = \left(\hat V(x) - \beta(x) \right) = L(x, \hat\pi(x)) + \gamma \left(\hat V(x^+) - \beta(x^+) \right). \tag{17}$$

This gives:

$$\beta(x) = -L(x, \hat\pi(x)) - \gamma \hat V(x^+) + \hat V(x) + \gamma \beta(x^+). \tag{18}$$

Note now that, by definition of $\hat\pi(x)$,

$$L(x, \hat\pi(x)) + \gamma \hat V(x^+) = \min_j \left(L(x, u_j) + \gamma \sum_{k=1}^N h_k(f(x, u_j)) V_k \right) \tag{19}$$

and, henceforth, the term $-L(x, \hat\pi(x)) - \gamma \hat V(x^+) + \hat V(x)$ is equal to the TDE, $\epsilon(x)$, defined in (12). □

The basic (informal) idea, then, is realizing that minimizing the temporal-difference error (TDE) $\epsilon(x)$ will minimize $\beta(x)$, because TDE accumulates along trajectories, in a similar way to the actual cost: a large $\epsilon(x)$ in one of the successors of a given x_0 will entail a large $\beta(x_0)$, of course taking into account the discount factor. The focus of the mesh adaptation should consider adding points, say x_{new}, at the places where $\epsilon(x_{new})$ is largest: adding them to $\mathbb{D}$ will automatically make zero its TDE after a new value iteration step. This will benefit the accuracy of the value function approximation of all *predecessors* of x_{new} under policy $\hat\pi(x)$: a large error in value function estimation accuracy at a particular point may be due to TDE at points far away from said point. This key idea will be refined in subsequent developments.

Lemma 2. *Consider the partial discretization MDP with continuous initial state proposed in Section 3.1. Let $x \in \mathbb{X}$ be an arbitrary point, not necessarily in the vertex dataset $\mathbb{D}$. The true value function of the said partial-discretization MDP, for every $x \in \mathbb{X}$ is:*

$$\hat V^*(x) := \min_j \left(L(x, u_j) + \gamma \sum_{k=1}^N h_k(f(x, u_j)) V_k \right). \tag{20}$$

Proof. Indeed, as $h_k(f(x,u))$ are the transition probabilities, (20) can be understood as

$$\hat V^*(x) = \min_j L(x, u_j) + \gamma E\left[\hat V^*(x^+) \right]$$

because V_k attain the optimal values at vertex points in $\mathbb{D}$, as their successors are also vertex points and, hence, the solution of (14), i.e., (15), is exact. □

The above lemma gives a new interpretation to the solution of (14), not only in terms of a MDP with a finite set of states, but in terms of an MDP where the initial condition lies in a continuous state space $\mathbb{X}$, even if its successor does lie in a finite set, later on. The referred solution is, in both cases, optimal. The optimal policy for the partial-discretization MDP arising from (20) will be denoted as $\hat\pi^*(x)$, $x \in \mathbb{X}$, i.e., the u_j minimizing the right-hand side of (20).

Corollary 1. *The Bellman temporal difference error $\epsilon(x)$ from (12) is coincident with the approximation error $\hat V(x) - \hat V^*(x)$.*

Proof is straightforward from definitions, omitted for brevity.

3.3. Variance of the partial-discretization MDP

The reinterpretation from barycentric coordinates to probabilities in the discretization allows us to obtain statistical descriptions of the resulting DP problems.

Lemma 3. *Let us denote the actual value of the cost (2) attained by a trajectory starting in $x \in \mathbb{X}$ with policy $\hat{\pi}^*(x)$ in the partial-discretization MDP as J_x; note that the latter is a random variable, whereas in the original continuous state–space (2) was deterministic (with deterministic $\pi(x)$).*

The expected quadratic error between the interpolated approximator $\hat{V}(x)$ and J_x will be denoted as $\delta^2(x) := E[(J_x - \hat{V}(x))^2]$. The expected quadratic error fulfills the following Bellman-like equation:

$$\delta^2(x) = \epsilon^2(x) + \nu(f(x, \hat{\pi}^*(x))) + \gamma^2 \sum_{k=1}^{N} h_k(f(x, \hat{\pi}^*(x)))\delta^2(x_k), \tag{21}$$

where $\nu(\xi)$, to be informally understood as 'immediate variance', is:

$$\nu(\xi) = \gamma^2 \sum_{k=1}^{N} h_k(\xi) \cdot \left(V_k - \sum_j h_j(\xi) V_j \right)^2. \tag{22}$$

Proof. As $E[J_x] = \hat{V}^*(x)$ by definition of the value function, we have that

$$\delta^2(x) = E[(J_x - \hat{V}(x))^2] = (\hat{V}^*(x) - \hat{V}(x))^2 + \sigma^2(x) \tag{23}$$

where $\sigma^2(x) := E[(J_x - \hat{V}^*(x))^2]$ denotes the variance of the random variable J_x. This is a well-known bias–variance decomposition. Corollary 1 allows then writing (23) as:

$$\delta^2(x) = \epsilon^2(x) + \sigma^2(x) \tag{24}$$

Now, we can realize that

$$J_x = L(x, \hat{\pi}^*(x)) + \gamma \zeta_x, \tag{25}$$

where random variable ζ_x is governed by mixture distribution selecting random variable J_{x_k} with probability $h_k(f(x, \hat{\pi}^*(x)))$. Hence, from standard formulae of the variance of mixture distributions, we can assert that the variance of ζ_x is:

$$\sigma^2_{\zeta_x} = \sum_{k=1}^{N} h_k(f(x, \hat{\pi}^*(x))) \left(\sigma^2(x_k) + \left(V_k - \sum_{j=1}^{N} h_j(f(x, \hat{\pi}^*(x))) V_j \right)^2 \right). \tag{26}$$

Thus,

$$\sigma^2(x) = \gamma^2 \sigma^2_{\zeta_x} = \gamma^2 \sum_{k=1}^{N} h_k(f(x, \hat{\pi}^*(x)))\sigma^2(x_k) + \nu(f(x, \hat{\pi}^*(x))) \tag{27}$$

Now, jointly with (24) and the fact that $\delta^2(x_k) = \sigma^2(x_k)$ for points $x_k \in \mathbb{D}$ because $\epsilon(x_k) = 0$, allows us to write (21). □

Note that, in order to explicitly compute $\delta^2(x)$ for $x \notin \mathbb{D}$ we need to first compute $\sigma^2(x_k)$ for points in $\mathbb{D}$. Using shorthand notation $\sigma_k := \sigma(x_k)$, we can write a Bellman-like equation only evaluated for points in $\mathbb{D}$:

$$\sigma_j^2 = \gamma^2 \sum_{k=1}^{N} h_k(f(x_j, \hat{\pi}^*(x_j)))\sigma_k^2 + \nu(f(x_j, \hat{\pi}^*(x_j))), \qquad j = 1, \dots, N \tag{28}$$

which can be solved by, say, value iteration, see variance computation in Munos and Moore (2002). Note that the partial-discretization MDP allows us, generalizing the cited work, to extend the definitions of $\sigma^2(x)$ and $\nu(x)$ to the whole continuous state space $\mathbb{X}$ instead of applying them only to vertex points in $\mathbb{D}$.

Analyzing expression (21), some interpretations can be made when returning to the original continuous state–space problem:

1. The "bias" term $\epsilon^2(x)$ is made zero if x is added to the dataset $\mathbb{D}$, because after VI, $\epsilon(x) = 0$ for all x in $\mathbb{D}$.
2. The "immediate variance" term $\nu(f(x, \hat{\pi}^*(x)))$ is made zero if $x^+ = f(x, \hat{\pi}^*(x))$ is added to the dataset $\mathbb{D}$, as in such a case $h(x^+)$ will consist on a zero vector except a single element equal to one; in this case, after VI has converged again, $\nu(x^+)$ will be equal to zero, from its definition (22).
3. The last term (cumulative variance of vertices of the simplex where successor lands) depends on the whole trajectories starting at said vertices. In principle, we will assume, as an approximation, that this long-term effect does not significantly change by inserting either x or x^+ onto the vertex dataset.

The above discussion will inspire choosing $\epsilon^2(\cdot)$ and $\nu(\cdot)$ as key elements in the mesh adaptation criterion in later sections. However, the idea needs to be further refined, in order to consider different needs for accuracy of the value function estimate in certain regions of interest, in a global sense.

3.4. Influence over a region of interest

As just hinted, in an ADP problem there may exist a given "region of interest" where increased accuracy of the value function estimate is needed, whereas accuracy in other regions may be less important. The said region of interest usually might be coincident with the region of initial conditions that are assumed to be more likely in practice. For instance, in later examples, we will consider of interest a neighborhood of the downwards equilibrium of an inverted pendulum, or the 'valley" equilibrium in a mountain-car setup.

In summary, a high-accuracy estimate over whole state space $\mathbb{X}$ might require a lot of vertex points in $\mathbb{D}$ but, if we assume that initial conditions lie in a known set Ω, for a given policy, then there will be regions of $\mathbb{X}$, not visited by the trajectories starting in Ω, where accuracy may be diminished with no deleterious effect on actual performance. For instance, in later examples, concentrating on a given circle of initial conditions will make adaptive meshing to focus only on regions covered by trajectories starting in said circle, see Fig. 7.

The initial condition set idea may be generalized to a initial-condition probability density function $\phi(x) : \mathbb{X} \mapsto R^+$. The formalization of the above ideas roots in the concept of *influence* (Munos and Moore, 2002), to be extended and adapted to our problem setting next.

3.4.1. Weighted influence

Given an initial-condition probability distribution with density $\phi(x)$, and a policy $\pi(x)$, the expected cost for repeated experiments will be

$$V^\phi := \int_{\mathbb{X}} \phi(x) V^\pi(x)\, dx$$

and the influence of a point $x_k \in \mathbb{D}$ on said V^ϕ, understood as $\frac{\partial V^\phi}{\partial L_k}$, with L_k as shorthand to $L(x_k, \pi(x_k))$, see Munos and Moore (2002, Eq. (8)), will be:

$$I^\phi(x_k) := \frac{\partial V^\phi}{\partial L_k} = \int_{\mathbb{X}} \phi(x) \frac{\partial V^\pi(x)}{\partial L_k}\, dx. \tag{29}$$

If, according to the assumptions in this paper, we use a polyhedral partition for function approximation, we will assume $V^\pi(x) = \sum_{j=1}^{N} h_j(x) \cdot V_j$, for some V_j, after approximate policy evaluation has been carried out. Thus, we have:

$$I^\phi(x_k) = \int_{\mathbb{X}} \phi(x) \sum_{j=1}^{N} h_j(x) \frac{\partial V_j}{\partial L_k}\, dx = \sum_{j=1}^{N} \left(\int_{\mathbb{X}} \phi(x) h_j(x)\, dx \right) \cdot \frac{\partial V_j}{\partial L_k}, \tag{30}$$

where $\frac{\partial V_j}{\partial L_k}$ coincides with the influence $I(\xi_k|\xi_j)$ defined in Munos and Moore (2002). We will denote $\beta_j := \left(\int_{\mathbb{X}} \phi(x) h_j(x)\, dx \right)$, so (30) can be rewritten as: $I^\phi(x_k) = \sum_{j=1}^{N} \frac{\partial V_j}{\partial L_k} \beta_j$.

Lemma 4. *Given an initial-condition probability distribution with density $\phi(x)$, and a policy $\pi(x)$, the influence of vertex points in $\mathbb{D}$, verifies:*

$$I^{\phi}(x_k) = \gamma \cdot \sum_{j=1}^{N} h_k(f(x_j, \pi(x_j)))I^{\phi}(x_j) + \int_{\mathbb{X}} \phi(x) h_k(x)\, dx \tag{31}$$

Proof. Following (Munos and Moore, 2002, section $8.4), the operator Γ_ξ defined in the cited work, on a finite dataset $\mathbb{D}$, is linear, and can be expressed as a matrix whose element at row k, column j are $\gamma h_k(f(x_j, \pi(x_j)))$; the whole k'th row of such matrix Γ_ξ represents the terms that multiply influences in $\gamma \cdot \sum_{j=1}^{N} h_k(f(x_j, \pi(x_j)))I^{\phi}(x_j)$ in (31). Then, Munos' work proves that the influence, defined as $I(\xi_k|\xi_j) := \frac{\partial V_j}{\partial L_k}$, coincides with the element at row k, column j of matrix $(I - \Gamma_\xi)^{-1}$. If we denote as $W := (\beta_1, \beta_2, \dots, \beta_N)^T$, then the solution of (31) is $(I - \Gamma_\xi)^{-1} \cdot W$, which coincides with the element at the right-hand side of (30). □

Note that, instead of inverting a (large) matrix, computation of $I^{\phi}(x_k)$ can be carried out iteratively, with the same cost as computing the value function of a discounted Markov chain, see Munos and Moore (2002) for details.

It is important to remark that Munos' equation (11) replaces β_k with just one or zero in (31), depending on x_k being inside or outside of the region of interest (in Munos' case, such region of interest is based on a policy disagreement criterion). This breaks the connection with (29), whereas our integral form of β_k adapts to the density of the dataset points in the region of interest, implicitly incorporating considerations about the "volume" of each point's local patch: our modification to Munos' proposal reduces the influence of a given point as meshing progresses if its local patch gets smaller; for instance, if $\phi(x)$ were a uniform distribution β_k would be proportional to the volume of the local patch of x_k.

3.4.2. Influence estimation for candidate new points

The developments in Section 3.4.1 compute the influence $I^{\phi}(x_k)$ of points x_k in the dataset $\mathbb{D}$. Therefore, if the value function estimate at a given point of $\mathbb{D}$ changes, it would be "propagated" to V^{ϕ} multiplied by said influence. However, value function estimates will change in adaptive meshing only when vertices are added or removed.

Therefore, if we now consider an arbitrary point x_{new} as a candidate to be added to the dataset $\mathbb{D}$, we might wish to approximate its "influence" over the average value function estimate V^{ϕ}. Clearly, the exact way of computing such influence would be modifying the triangulation and recomputing (31). However, that has the same cost as a value iteration step, so the computational cost of repeating it for every candidate point is unaffordable, as many of them may be evaluated at each mesh refinement step, as discussed in next subsection.

Our proposed approximation, then, consists on freezing the influences of points already in $\mathbb{D}$ and solving (31) in Lemma 4 only for the influence of x_{new}, assuming a re-triangulation of the simplex $\mathcal{S}$ in which x_{new} lies, see Fig. 2. Evaluating (31) for x_{new} considering $N+1$ points (those in $\mathbb{D}$ plus x_{new}) results in:

$$\begin{aligned} I^{\phi}(x_{new}) &= \gamma \hat{h}_{new}(f(x_{new}, \pi(x_{new})))I^{\phi}(x_{new}) \\ &+\gamma \cdot \sum_{j=1}^{N} \hat{h}_{new}(f(x_j, \pi(x_j)))I^{\phi}(x_j) + \int_{\mathbb{X}} \phi(x) \hat{h}_{new}(x)\, dx, \end{aligned} \tag{32}$$

i.e.:

$$I^{\phi}(x_{new}) = \frac{\gamma \cdot \sum_{j=1}^{N} \hat{h}_{new}(f(x_j, \pi(x_j)))I^{\phi}(x_j) + \int_{\mathbb{X}} \phi(x)\hat{h}_{new}(x)\, dx}{1 - \gamma \hat{h}_{new}(f(x_{new}, \pi(x_{new})))}, \tag{33}$$

being $\hat{h}_{new}(\cdot)$ the component associated to x_{new} of the barycentric coordinates of successor points in the modified triangulation. Obviously, in actual implementation, summation in (33) needs to be carried out only for these j such that $f(x_j, \pi(x_j))$ belongs to the re-triangulated simplex $\mathcal{S}$ and, likewise, denominator would be unity if $f(x_{new}, \pi(x_{new})) \notin \mathcal{S}$, details left to the reader.

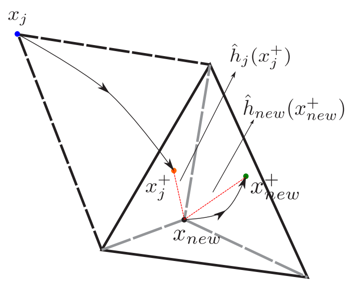


Fig. 2. Conceptual sketch for influence computation in Eq. (33); in the figure, $x_{new}^+ = f(x_{new}, \pi(x_{new}))$, and $x_j^+ = f(x_j, \pi(x_j))$. The original simplex $\mathcal{S}$, triangle in solid line, is retriangulated after adding x_{new} (dashed gray lines).

3.5. Summary: proposed criterion for point addition in adaptive meshing

The previous section has detailed how to obtain the expected quadratic error of the value function approximation at an arbitrary point $x \in \mathbb{X}$, see (21) in Lemma 3, as well as influence of such error over a region of interest (formally, given a probability distribution of initial conditions, over an average value function V^{ϕ}), see (33) derived from Lemma 4.

Our proposed criterion to sort a set of candidate points is

$$C(x) = (I^{\phi}(x))^2 \cdot (\epsilon^2(x) + \nu(x)). \tag{34}$$

Indeed, if x is added to the dataset $\mathbb{D}$, both the "bias" term $\epsilon^2(x)$ and the "immediate variance" term $\nu(x)$ are made zero after a new value iteration step; note that points already in $\mathbb{D}$ will render a zero value of $C(x)$. As influence has a gradient interpretation, the expected quadratic error in a point x would propagate to V^{ϕ} multiplied by the square of such influence.

Therefore, in summary, our proposal consist in asserting that the point (or points, if so wished) with largest $C(x)$ should be added to $\mathbb{D}$ and value iteration plus influence computations repeated for subsequent mesh refinement steps.

Of course, finding the maximizer of the criterion in all of the state space $\mathbb{X}$ may be cumbersome; in actual implementations of our algorithms in Section 4, the criterion is evaluated only in a number of random points in $\mathbb{X}$, plus the barycenters of the simplicial cells plus points close to the edge's midpoints (one barycentric coordinate equal to zero), inspired in Grüne (1997). The random exploration helps in finding "bad" transitions particularly in initial iterations where the starting volume of partition cells may be large.

Intuitive interpretation of the refinement criterion. Our proposal gives a sensible justification of polyhedral splitting based on:

1. Large Bellman error $\epsilon^2(x)$, accumulating value function estimation errors on the predecessors of x, Lemma 1.
2. Large value of $\nu(x)$, Lemma 3, Eq. (22); this detects, for instance, points near the barycenter of simplices with large differences between the value function in the vertices.
3. Modulating the above factors with the influence over the average cost V^{ϕ} with a given probability distribution of initial conditions, Lemma 4: (*a*) simplices with large volume in regions of high-probability initial states have points with large influence, because these points yield a large value of $\int_{\mathbb{X}} \phi(x)\hat{h}_{new}(x)\, dx$ in (33), (*b*) If we consider points such that many trajectories (starting in high-probability initial states) pass near them, these will also be influential points.

Note that (34) might, at first glance, seem reminiscent of the square of Munos' criterion `Std·Influence` (Munos and Moore, 2002) method, but neither influence nor standard deviation are computed in the way we are proposing in this work. Comparative discussion will be provided at the end of the example section.

4. Simulation and experimental results

This section will present examples on well-known benchmarks (mountain-car and inverted pendulum), and compare our adaptive meshing criterion with proposals in earlier literature. At the end of the section, we show an experimental result on the application of the proposed method on a robot arm and also discuss about it.

A "reference" value function and associated (approximate) optimal controller is obtained by setting up a 51 × 51 uniform grid on the state space and a Delaunay triangulation. The goal of the examples will be improving over such "dense" uniform grid with a significantly lower number of vertex points in $\mathbb{D}$. The performance figures will be computed by adding up the cost of all trajectories starting from the 2601 validation-set points of the same 51x51 grid, obtained by actual *simulation*, so there is no "approximation" in that policy evaluation step. The benchmark performance values in later plots (Figs. 4 and 9) will be the cost difference between the above "reference" 2601-point controller and the one arising from alternative meshing refinement options:

1. Regular grid meshing (11 × 11, 21 × 21, 31 × 31, 41 × 41 and 51 × 51);
2. $|TDE|$·Area (Armesto and Sala, 2022), based on Grüne's work (Grüne, 1997) with a minor modification of the criterion that multiplies the TD error ϵ by the area of the simplex, to avoid getting stuck at discontinuities as pointed out by Munos and Moore (2002);
3. Munos `Std·Influence` method, given that it is the one that Munos and Moore (2002) finally recommends after an extensive comparative analysis;
4. Our proposed selection criterion (34).

Thus, reaching the value of zero performance in said plots will mean equaling the validation-set performance of the densest of the tried grids, but with a lower number of mesh points.

In order to obtain comparable results between different methods, all mesh refinement steps in the four above options have been implemented by splitting a triangle by its edge mid-points, and adjacent triangles are also divided to ensure that the value function is continuous (see Grüne (1997, Fig. 1)). For brevity and clarity of exposition, no point removal criterion has been implemented in the comparisons.

4.1. Example 1: Mountain-car

Let $x = [p\ v]^T$ be the state of the mountain-car system, see Fig. 3, Knox et al. (2011), being p the position of the car and v its velocity and with the following dynamics $\dot{p} = v$ and $\dot{v} = -g\sin(ap+b)+u$, being g the gravity magnitude; $a = \frac{3\pi}{80}\text{m}^{-1}$ and $b = \frac{\pi}{80}$ define the sinusoidal shape of the mountain and u is the control action expressed in m/s^2. The state $x = (p, v)$ is limited to the range $x \in \mathbb{X} = [-10, 10] \times [-15, 15]$, while the control action is assumed to be in the discrete set $u \in \mathbb{U} = \{-4, -\frac{8}{3}, -\frac{4}{3}, 0, \frac{4}{3}, \frac{8}{3}, 4\}$ m/s^2. The immediate cost is set to $L(x, u) = 6$ and discount factor is $\gamma = 0.99$. The penalization or terminal cost for being outside the region $\mathbb{X}$ is 4000, except when a target set, defined as $p \geq 10$ and $v \geq 0$, is reached: in that case terminal cost is 0, see Fig. 3; for convenience in implementation steps, the state has been normalized so that x maps into a normalized region $\tilde{\mathbb{X}} = [0, 1] \times [0, 1]$, so from now on, the state is expressed in normalized coordinates. Thus, the goal is to reach the normalized position $\tilde{p} \geq 1$, $\tilde{v} \geq 0.5$.

Fig. 4 shows the performance of the previously enumerated meshing methods as the number of points increases. In this example, the probability distribution of initial conditions $\phi(x)$ in Section 3.4.1 was set to a constant (uniform distribution) in our proposal.

As it can be seen in the figure, the performance of $|TDE|$·Area method and our criterion are even better than the 2601-point reference one (zero is reached by the $|TDE|$·Area method after 700 points, our proposal here reaches zero after 200 points). Munos' method starts with

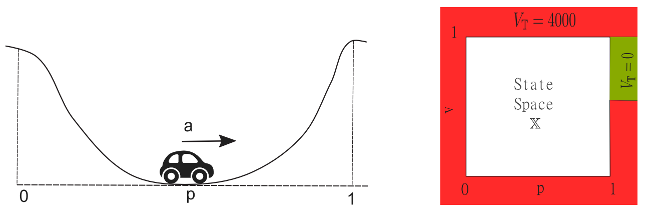


Fig. 3. Mountain Car representation (left) and terminal set definition (normalized coordinates, right).

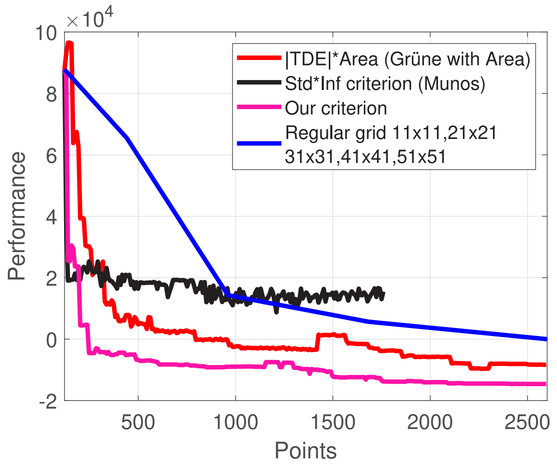


Fig. 4. Mountain-Car benchmark comparative results. Zero indicates the same performance level as a regular grid with 2601 points.

a sharp improvement but gets quickly stuck and ceases improving the performance of the resulting mesh.

In Fig. 5, the value function of our proposal is shown (color map), as well as the vertex points in $\mathbb{D}$ (black ×). In addition to this, the figure shows the phase plot of trajectories starting at normalized points $[0.4, 0.5]^T$, $[0.5, 0.5]^T$, $[0.6, 0.5]^T$, $[1, 0.5]^T$, $[1, 0.4]^T$, $[1, 0.3]^T$, $[1, 0.2]^T$, $[1, 0.12]^T$, $[1, 0.1]^T$. These points where selected so that the car starts, in the first three of them, close to the bottom of the mountain at zero speed – 0.5 after normalization –, or it starts close to the top of the mountain, but with a negative velocity (last six). The last of the points, for instance, has a too large initial (negative) speed so that overrunning the left extreme ($\tilde{p} < 0$) is unavoidable because of 'braking' saturation.

Fig. 6 shows the control map derived from such value function that has been used to simulate such trajectories. The figure shows that our adaptive meshing tends to concentrate points along the value function discontinuities for better approximation accuracy, as intuitively expected.

If, instead of a uniform initial condition distribution ϕ we considered $\phi(x)$ as a uniform distribution in a circle, the adaptive meshing would progress as shown in Fig. 7, plots (b) and (c); the effect of "influence" is clear, as the mesh points are concentrated along the trajectories emanating from said circles: all the effort (see Fig. 5) in refining accuracy at the discontinuities, at the left of said Fig. 7, ceases being relevant in Fig. 7(b,c) for the given initial condition sets.

4.2. Example 2: Inverted pendulum

Let $x = [\alpha\ \dot{\alpha}]^T$ be the state of an inverted pendulum, being α the joint angle ($\alpha = 0$ represents the unstable upward configuration) and $\dot{\alpha}$ the joint angular velocity. Dynamics is $\ddot{\alpha} = a\sin(\alpha)+bu$, being $a = 42.269\text{s}^{-2}$, $b = 24.206\text{Kg}^{-1}\text{m}^{-2}$ and u is the torque applied to the joint. The state is limited to $x \in \mathbb{X} = [-\pi, \pi] \times [-10\pi, 10\pi]$. The goal is optimizing a discounted quadratic cost with immediate cost $L(x, u) = x^T Qx + u^T Ru$,

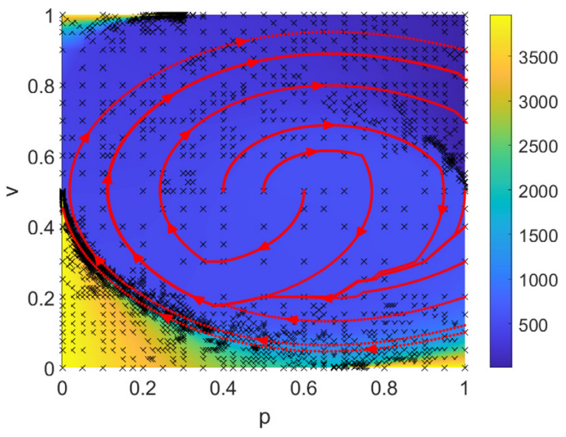


Fig. 5. Phase Plot of optimal mountain-car trajectories from different initial states.

with $Q = \mathrm{diag}\{10, 0.1\}$, $R = 1$ and discount factor $\gamma = 0.999$. As in the previous example, the state space will be normalized to $[0, 1] \times [0, 1]$.

Note that, if the state is close to the (unstable) equilibrium point, we may approximately solve the optimal control problem via linearization, or, for a non-infinitesimal region around said equilibrium, with sector-nonlinearity models and linear matrix inequalities (LMIs), Tanaka and Wang (2001) and Robles et al. (2019). Hence, as we have efficient computational tools to obtain a solution in said region, we will introduce a *terminal ellipsoid* around the origin, with an LMI-based *terminal controller* and associated *terminal cost*. This avoids wasting resources in a region where a quasi-optimal solution is well known. The LMIs and constraints involved in the specification of the terminal ellipsoid and terminal controller are detailed in Appendix.

As the terminal ellipsoid is determined considering the saturation bound of the terminal control law, intuitively, the optimal controller with continuous u will be saturating in most of the state space far away from the origin (outside the terminal ellipsoid). For that reason, we will assume that, outside the terminal ellipsoid, the control action will be saturated, taking only two possible values $u \in \mathbb{U} = \{-1, 1\}$ Nm, in order to speed up dynamic programming computations with negligible performance loss.

As discussed at the start of the example section, the reference design to compare with is a regular 51 × 51 grid. However, to properly take terminal ingredients into account, points in the grid inside the terminal ellipsoid have been removed from such grid, and 16 points in the boundary of said ellipsoid have been added (the green points in Fig. 8); these ellipsoid points have been added in all tested *regular grid* meshes.

In this second example, initial conditions have been assumed to lie in a circle of radius 0.05 (in normalized coordinates) around the bottom stable equilibrium (the goal is to swing the pendulum up and stabilize its position around $\alpha = 0$'). All adaptive meshing algorithms have been initialized with an 11 × 11 regular grid plus terminal ellipsoid (green) and 26 initial condition points (cyan), see Fig. 8.

Fig. 9 shows a benchmark of the performance error (difference with respect to the performance of the 51 × 51 regular grid) when simulations are started at the above-referred set of 26 initial condition points close to the bottom equilibrium (instead of simulating the whole 2601 points as in Example 1). As it can be seen in the figure, the proposed criterion, in solid magenta line, compares well against the other alternative options in consideration: it obtains the best performance of all methods after adding 2601 points. Fig. 10 shows the final control map and a perspective view of the estimated value function. Note that, as the objective is a good estimation along trajectories starting at cyan points in Fig. 8. Indeed, Fig. 10(b) does not necessarily capture the features of the value function over the whole state space: this is *intentional*, to avoid adding unnecessary points, which can be seen as

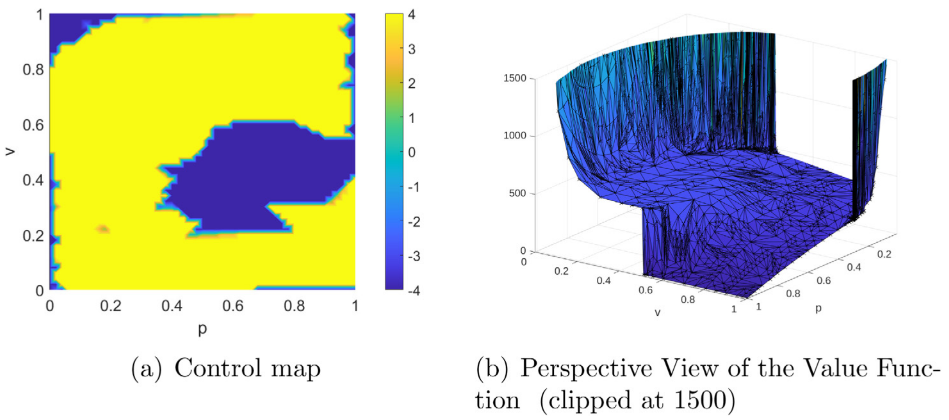


Fig. 6. Control Map associated to the Value Function (mountain-car).

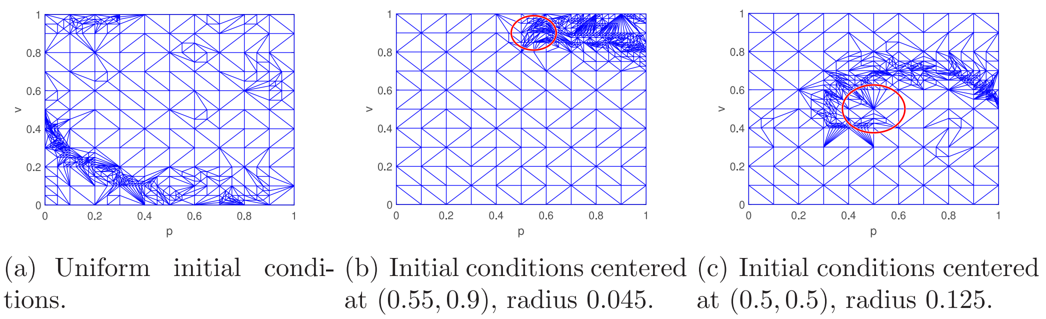


Fig. 7. Evolution of the Mountain-car problem's triangulation when reaching 500 points under various initial condition influence settings (uniform in all state space versus uniform in small circles).

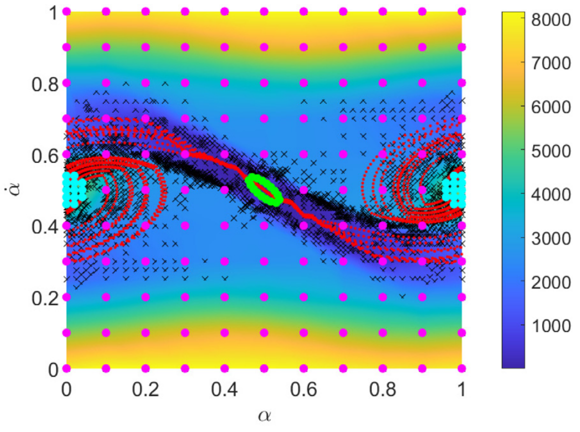


Fig. 8. Example 2 (inverted pendulum): value function with phase plot trajectories from 26 initial conditions (cyan dots). Magenta+ green+ cyan dots represent initial mesh points, while black crosses indicate the additional points that conform the final mesh. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

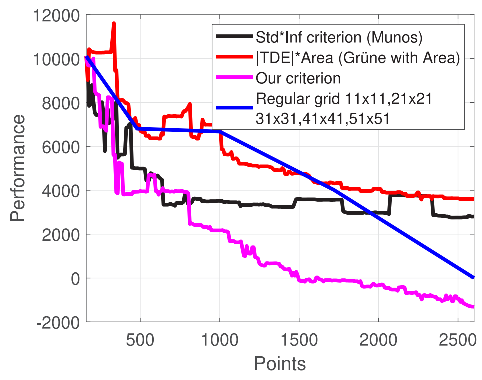


Fig. 9. Inverted Pendulum benchmark comparative performance. Zero indicates the same performance level as a regular grid with 2601 points.

an advantage of our proposal (no points at high velocities – away from 0.5 – have been added apart from those in the initial 11 × 11 grid).

4.3. Experimental results

The proposed method will be experimentally validated using the low-cost robot arm meArm (UPV version) (Armesto, 2021a) depicted in Fig. 11. Kinematics of this particular robot is governed by:

$$\dot{q}(t) = J(q(t)) \cdot u(t), \tag{35}$$

where $q(t)$ is the robot configuration (joint angles), $J(q(t))$ is the end-effector robot Jacobian matrix and $u(t)$ is the control action (joint velocities); see Armesto (2021b) for details.

The robot has three degrees of freedom (DoF), one for rotating the wrist of the robot that allows to point to a specific direction, while the other two DoF form a parallelogram structure that controls the height and displacement of the robot gripper. Thus, the first DoF is used to point towards an object, while the other two DoFs will be used to place the gripper at the correct position, simplifying the problem to a coplanar case.

We are interested in the coplanar motion planning problem, so the aim is to provide references for the second and third joints, here denoted as q_2 and q_3 respectively, conforming a state $x := [q_2\ q_3]^T$ for the dynamic programming setup.

The goal of the experiment is driving the robot from *home* configuration $x_0 := [\frac{\pi}{2}\ \frac{\pi}{2}]^T$ to a *goal* configuration $x_T := [3.009\ \frac{\pi}{2}]^T$, without colliding with the box, floor or even itself (see Fig. 11), in minimum time.

We have reproduced the experimental setup on a simulation environment based on CoppeliaSim (Rohmer et al., 2013) where a sweep on the configuration space $\mathbb{X} = [\frac{85\pi}{180}, \pi] \times [\frac{20\pi}{180}, \pi]$ has been carried out to precompute which configurations are free of collisions, usually denoted as $C_{free} \subset \mathbb{X}$. The aim is optimizing a discounted minimum time problem with immediate cost defined as:

$$L(x,u) = \begin{cases} 1 & x \in C_{free} \\ 4000 & \text{otherwise} \end{cases}, \tag{36}$$

with $\gamma = 0.999$ and as terminal cost $V_{\mathbb{T}}(x) = 4000$ if $x \notin \mathbb{X}$, plus the *goal* configuration that takes $V_{\mathbb{T}}(x_T) = 0$. In summary, exiting the allowed configuration space or colliding with objects is penalized with a cost of 4000 units.

Control actions are joint speeds, defined with a discrete set as $\mathbb{U} = [-\frac{50\pi}{180}, \frac{50\pi}{180}] \times [-\frac{50\pi}{180}, \frac{50\pi}{180}]$ with a regular grid of 5 actions on each dimension.

Fig. 12(a) shows the results of our adaptive meshing algorithm, obtained after applying the proposed method to an initial regular grid of 9 × 9 of configurations plus the *goal* state (82-point initial mesh, Delaunay triangulated). We have depicted in red the configuration obstacle set[1], that is, $C_{obs} = \mathbb{R}^2 \setminus C_{free}$. As it can be seen, the mesh after 500 points specializes the partitions along the optimal trajectory between the *home* configuration (in normalized coordinates corresponds to the point $\tilde{x}_0 = [0.0526\ 0.4375]^T$ depicted as a magenta point in the Figure) and the *goal* configuration ($\tilde{x}_T = [0.92\ 0.44]^T$ in normalized coordinates depicted as a green point in the Figure). The initial-condition region is depicted with a magenta circle of radius 0.05 in normalized coordinates around the point $\tilde{x}_0$. In Fig. 12(b) we have depicted the state and control action trajectories obtained after applying the resulting controller, while images shown in Fig. 11 are frames of the experiment performed with the actual robot.

5. Discussion

The two above-discussed examples show good performance of our proposal compared to prior art. The main reasons of this, supported by the lemmas in Section 3, are:

Improved behavior in discontinuities. Local TDE-only methods can get caught in discontinuities (or sharp transitions) of the value function. Indeed, simplicial meshes provide a continuous function approximator, thus, they cannot capture discontinuities so they keep adding points at very close places; this was already pointed out in Munos and Moore (2002). This is the motivation of including an heuristic "area" modification in our preliminary work (Armesto and Sala, 2022), to avoid endless splitting of conflictive simplices. Using other continuous function approximators (neural networks, …) would in principle share this inherent discontinuity problem.

Improved adaptation to given initial conditions. A way to improve over the "area" heuristic in prior works is the weighted influence proposed here, as shown in performance plots. The influence idea was initially proposed in Munos and Moore (2002), but without weights (only point-counting), and using as source states the points with control-map discontinuities. In this sense, we refined Munos' ideas in Section 3.4 considering the probability distribution of initial conditions $\phi(x)$ and the approximation of the influence of *new* candidate points to be added. In summary, improvements come because influence provides a "global" perspective of the relevance of a point, considering high-likelihood trajectories instead of "local" features (error, smoothness, …).

[1] box walls have been enlarged 2 mm so that when the robot passes close by the obstacle region it does not collide with it.

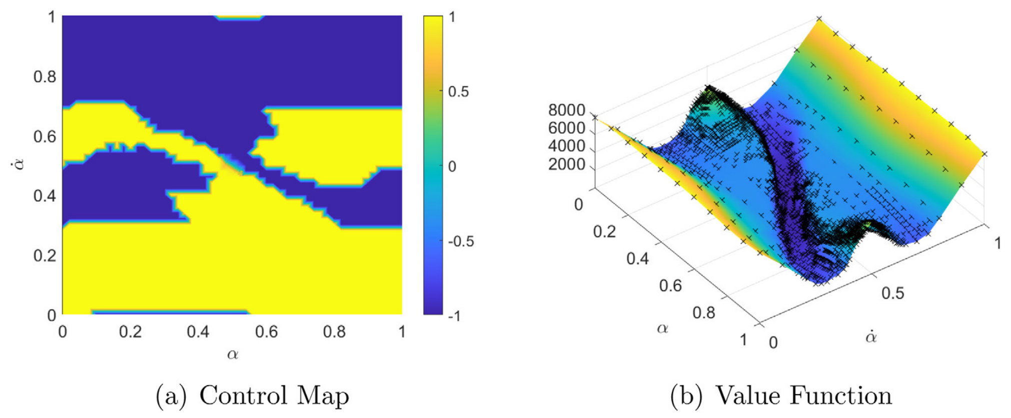
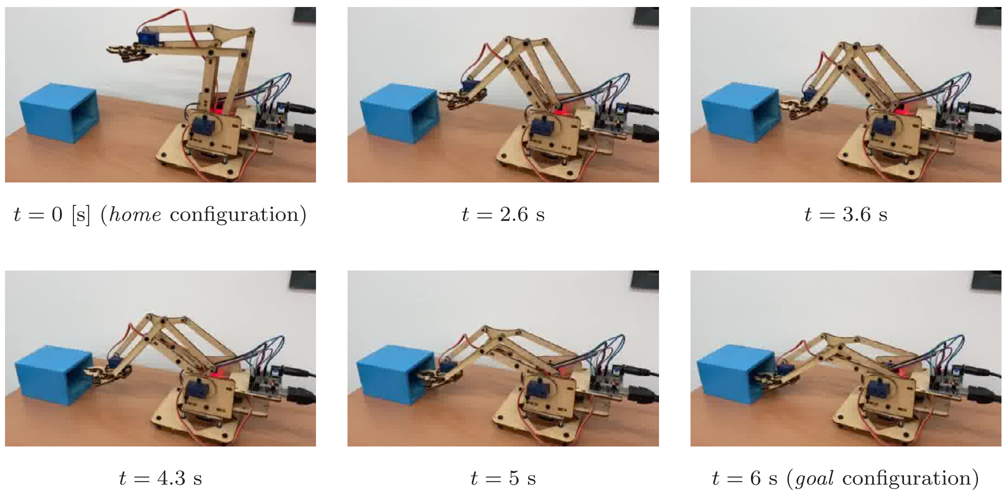


Fig. 10. Control map and value function for the inverted pendulum example.



$t = 0$ [s] (*home* configuration)
$t = 2.6$ s
$t = 3.6$ s
$t = 4.3$ s
$t = 5$ s
$t = 6$ s (*goal* configuration)

Fig. 11. meArm robot experimental results: sequence of frames, being *home* configuration at top-left, and *goal* configuration at bottom-right figures.

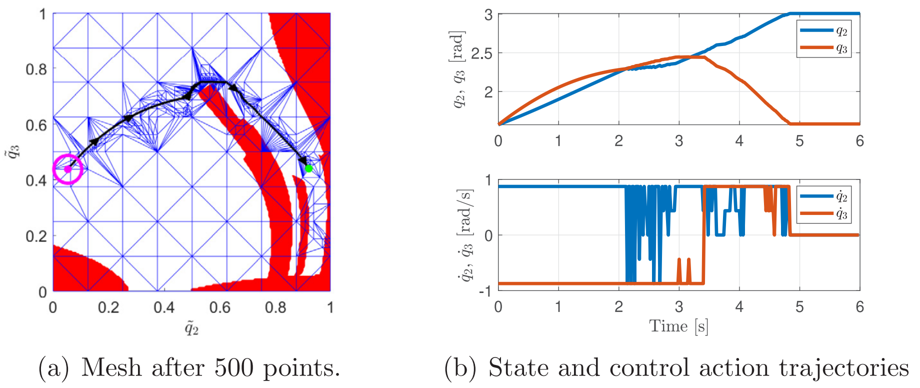


Fig. 12. Results of our proposed algorithm and meArm robot simulation. The task is moving the robot from *home* configuration $\tilde{x}_0 = [0.0526\ 0.4375]^T$, magenta dot, to *goal* configuration $\tilde{x}_T = [0.92\ 0.44]^T$, green dot, without colliding with the box, the table and itself, in minimum time. The obstacle mapping to the configuration space appears in red and the initial conditions region is depicted with a magenta circle around the point $\tilde{x}_0$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Improved estimation of error bias/variance propagation. Our criterion roots on the propagation of quadratic error in Lemma 3. Again, Munos and Moore (2002) computes an "Standard Deviation" figure based on the "variance" at database points, solution of (28). However, in the same way as the TDE accumulates to yield value function estimation errors (Lemma 1), a large value of such variance depends on the accumulation of "immediate variance" $v(x)$ along trajectories, defined in (22), so v is the quantity that is reduced by splitting a simplex, instead of σ_k^2 in the cited work. In fact, Lemma 3 shows that the sum of squared TDE plus immediate variance $v(x)$ is what actually needs to be considered in the simplex-splitting criterion.

The third remark, jointly with the weighted-influence consideration (second remark) is what builds up our proposal, with a clearer theoretical justification and good comparative performance over prior proposals.

6. Conclusions

This paper has proposed a criterion for mesh refinement in approximate dynamic programming using polyhedral partitions. A criterion for addition of points to a mesh is proposed, based on a series of lemmas regarding error propagation in approximate dynamic programming using polyhedral partitions as underlying piecewise-linear function approximator.

The paper discusses the reasons on why the proposed ideas yield performance improvement compared to prior literature, combining quadratic error propagation with the introduction of influence over initial condition probability distribution.

A couple of academic examples show good performance of the proposed criterion compared to alternative options, and a robotic experiment has been also provided showing good performance in a motion-planning problem.

Future research will combine these ideas with mesh coarsening options and efficient implementation, in order to apply these ideas to higher-dimensional problems.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Authors are grateful to the financial support of grant PID2020-116585GB-I00 funded by MCIN/AEI/10.13039/501100011033, Agencia Estatal de Investigación (Spanish government).

Appendix. LMI-based terminal ingredients

The inverted pendulum in Example 2 is modeled as a Takagi–Sugeno system $x_{k+1} = \sum_{i=1}^{2} \mu_i(x_k)A_i x_k + B_i u_k$, via sector non-linearity in a range of angular positions, assuming $|\alpha| \le \alpha_{max}$, with $\alpha_{max} = \frac{20\pi}{180}$. Using Euler discretization, we have model vertices as:

$$A_1 = \begin{pmatrix} 1 & T \\ aT & 1 \end{pmatrix} \qquad B_1 = \begin{pmatrix} 0 \\ bT \end{pmatrix} \tag{A.1}$$

$$A_2 = \begin{pmatrix} 1 & T \\ a\frac{\sin(\alpha_{max})}{\alpha_{max}}T & 1 \end{pmatrix} \qquad B_2 = \begin{pmatrix} 0 \\ bT \end{pmatrix}. \tag{A.2}$$

Any policy $u = \pi(x)$ and value function estimate $\overline{V}^{\pi}$ satisfying the following inequality:

$$\overline{V}^{\pi}(x) \ge L(x, \pi(x)) + \gamma \overline{V}^{\pi}(f(x, \pi(x))) \tag{A.3}$$

is a policy with guaranteed cost, i.e., the actual value function of the policy fulfills $V^{\pi}(x) \le \overline{V}^{\pi}(x)$. If we enforce a quadratic expression $\overline{V}^{\pi}(x) := x^T P^{-1} x$, and control law $\pi(x) = -\sum_{i=1}^{2} \mu_i(x) F_i P x$, then a sufficient condition for (A.3), using Tuan relaxation (Tuan et al., 2001), are the following LMIs (Tanaka and Wang, 2001; Díaz et al., 2020) in decision variables P, F_1, F_2:

$$LMI(i,j) := \begin{pmatrix} P & P & F_j & \sqrt{\gamma}(A_i P - B_i F_j)^T \\ P & Q^{-1} & 0 & 0 \\ F_j & 0 & R^{-1} & 0 \\ \sqrt{\gamma}(A_i P - B_i F_j) & 0 & 0 & P \end{pmatrix} \tag{A.4}$$

$$0 \le LMI(1,1) \tag{A.5}$$

$$0 \le LMI(1,1) + LMI(1,2) + LMI(2,1) \tag{A.6}$$

$$0 \le LMI(2,2). \tag{A.7}$$

In this case, maximizing the trace of P, we obtain:

$$K \equiv K_1 = K_2 = [4.7245\ 0.6717] \tag{A.8}$$

$$P^{-1} = \begin{pmatrix} 259.94 & 21.93 \\ 21.93 & 3.08 \end{pmatrix}. \tag{A.9}$$

The ellipsoid $\{x : x^T P^{-1} x \le V_{max}\}$ defines the terminal ellipsoid considered in the example, where V_{max} is obtained via another LMI in decision variable V_{max}, to be maximized, with the following two inequalities:

$$P^{-1} - \frac{K^T K}{u_{max}^2} V_{max} \ge 0, \qquad P^{-1} - \begin{pmatrix} \frac{1}{\alpha_{max}^2} & 0 \\ 0 & 0 \end{pmatrix} V_{max} \ge 0, \tag{A.10}$$

as a consequence of imposing that the control action must be smaller or equal than its maximum admissible value $|u| \le u_{max}$ and the angle of the pendulum must be inside the interval $|\alpha| \le \alpha_{max}$.

References

Altan, A., Hacıoğlu, R., 2020. Model predictive control of three-axis gimbal system mounted on UAV for real-time target tracking under external disturbances. Mech. Syst. Signal Process. 138, 106548.

Antos, A., Munos, R., Szepesvári, C., 2007. Fitted Q-iteration in continuous action-space MDPs. Adv. Neural Inf. Process. Syst. 20.

Antos, A., Szepesvári, C., Munos, R., 2006. Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. In: International Conference on Computational Learning Theory. Springer, pp. 574–588.

Armesto, L., 2021a. meArm robot (UPV version). https://www.thingiverse.com/thing:4589531. (Accessed 15 September 2021).

Armesto, L., 2021b. YouTube: Robotics systems (principles and foundations). https://www.youtube.com/playlist?list=PLjzuoBhdtaXOyB5ufadoxEfdsLfAFbSjd. (Accessed 15 September 2021).

Armesto, L., Sala, A., 2022. Volume-weighted Bellman error method for adaptive meshing in approximate dynamic programming. Rev. Iberoam. Autom. Inform. Ind. (in press).

Bertsekas, D.P., 2017. Dynamic Programming and Optimal Control, Vol. 1, fourth ed. Athena Scientific, Belmont, MA, USA.

Bertsekas, D.P., 2018. Abstract Dynamic Programming. Athena Scientific Nashua, NH, USA.

Busoniu, L., Babuska, R., De Schutter, B., Ernst, D., 2010. Reinforcement Learning and Dynamic Programming Using Function Approximators. CRC Press, Boca Raton, FL, USA.

Busoniu, L., de Bruin, T., Tolic, D., Kober, J., Palunko, I., 2018. Reinforcement learning for control: Performance, stability, and deep approximators. Annu. Rev. Control 46, 8–28.

Buşoniu, L., Páll, E., Munos, R., 2018. Continuous-action planning for discounted infinite-horizon nonlinear optimal control with Lipschitz values. Automatica 92, 100–108.

Cervellera, C., Macciò, D., 2017. A novel approach for sampling in approximate dynamic programming based on F-discrepancy. IEEE Trans. Cybern. 47 (10), 3355–3366.

Cohen, A., Dyn, N., Hecht, F., Mirebeau, J.-M., 2012. Adaptive multiresolution analysis based on anisotropic triangulations. Math. Comp. 81 (278), 789–810.

Davies, S., 1996. Multidimensional triangulation and interpolation for reinforcement learning. Adv. Neural Inf. Process. Syst. 9, 1005–1011.

De Farias, D.P., Van Roy, B., 2003. The linear programming approach to approximate dynamic programming. Oper. Res. 51 (6), 850–865.

Díaz, H., Armesto, L., Sala, A., 2020. Fitted Q-function control methodology based on Takagi-Sugeno systems. IEEE Trans. Control Syst. Technol. 28 (2), 477–488.

Díaz, H., Sala, A., Armesto, L., 2020. A linear programming methodology for approximate dynamic programming. Int. J. Appl. Math. Comput. Sci. 30 (2).

Farquhar, G., Gustafson, L., Lin, Z., Whiteson, S., Usunier, N., Synnaeve, G., 2020. Growing action spaces. In: International Conference on Machine Learning. PMLR, pp. 3040–3051.

Gomez Plaza, M., Arribas Navarro, T., Sanchez Prieto, S., 2017. Introducing MultiScale technique with CACM-RL. Int. J. Adv. Robot. Syst. 14 (1), 1729881417694289.

Gottesman, O., Futoma, J., Liu, Y., Parbhoo, S., Celi, L., Brunskill, E., Doshi-Velez, F., 2020. Interpretable off-policy evaluation in reinforcement learning by highlighting influential transitions. In: International Conference on Machine Learning. PMLR, pp. 3658–3667.

Grüne, L., 1997. An adaptive grid scheme for the discrete Hamilton-Jacobi-Bellman equation. Numer. Math. 75 (3), 319–337.

Grüne, L., Semmler, W., 2004. Using dynamic programming with adaptive grid scheme for optimal control problems in economics. J. Econom. Dynam. Control 28 (12), 2427–2456.

Keller, P.W., Mannor, S., Precup, D., 2006. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In: Proceedings of the 23rd International Conference on Machine Learning. pp. 449–456.

Knox, W.B., Setapen, A.B., Stone, P., 2011. Reinforcement learning with human feedback in mountain car. In: 2011 AAAI Spring Symposium Series.

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

Liu, F., Hager, W.W., Rao, A.V., 2015. Adaptive mesh refinement method for optimal control using nonsmoothness detection and mesh size reduction. J. Franklin Inst. B 352 (10), 4081–4106.

Liu, F., Hager, W.W., Rao, A.V., 2017. Adaptive mesh refinement method for optimal control using decay rates of Legendre polynomial coefficients. IEEE Trans. Control Syst. Technol. 26 (4), 1475–1483.

Maei, H.R., Szepesvari, C., Bhatnagar, S., Precup, D., Silver, D., Sutton, R.S., 2009. Convergent temporal-difference learning with arbitrary smooth function approximation. In: NIPS. pp. 1204–1212.

Maggipinto, M., Susto, G.A., Chaudhari, P., 2020. Proximal deterministic policy gradient. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS, IEEE, pp. 5438–5444.

Munos, R., Moore, A., 2002. Variable resolution discretization in optimal control. Mach. Learn. 49 (2–3), 291–323.

Munos, R., Szepesvári, C., 2008. Finite-time bounds for fitted value iteration. J. Mach. Learn. Res. 9 (5), 815–857.

Robles, R., Sala, A., Bernal, M., 2019. Performance-oriented quasi-LPV modeling of nonlinear systems. Internat. J. Robust Nonlinear Control 29 (5), 1230–1248.

Rohmer, E., Singh, S., Freese, M., 2013. CoppeliaSim (formerly V-REP): a versatile and scalable robot simulation framework. In: Proc. of the International Conference on Intelligent Robots and Systems. IROS.

Sherstov, A.A., Stone, P., 2005. Function approximation via tile coding: Automating parameter choice. In: International Symposium on Abstraction, Reformulation, and Approximation. Springer, pp. 194–205.

Tanaka, K., Wang, H.O., 2001. Fuzzy Control Systems Design and Analysis: A Linear Matrix Inequality Approach. John Wiley & Sons.

Timmer, S., Riedmiller, M., 2007. Fitted Q-iteration with CMACS. In: 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning. IEEE, pp. 1–8.

Tuan, H., Apkarian, P., Narikiyo, T., Yamamoto, Y., 2001. Parameterized linear matrix inequality techniques in fuzzy control system design. IEEE Trans. Fuzzy Syst. 9 (2), 324–332.

Whiteson, S., 2006. Evolutionary function approximation for reinforcement learning. J. Mach. Learn. Res. 7.

Whiteson, S., Taylor, M.E., Stone, P., 2007. Adaptive Tile Coding for Value Function Approximation. Tech. Report. AI-TR-07-339, Univ. of Texas at Austin.

Yershov, D.S., Frazzoli, E., 2015. Asymptotically optimal feedback planning: FMM meets adaptive mesh refinement. In: Algorithmic Foundations of Robotics XI. Springer, pp. 695–710.