Robotics: Science and Systems 2020
Corvalis, Oregon, USA, July 12-16, 2020

Kernel Taylor-Based Value Function Approximation for Continuous-State Markov Decision Processes

Junhong Xu†, Kai Yin†, Lantao Liu

Abstract—**We propose a principled kernel-based policy iteration algorithm to solve the continuous-state Markov Decision Processes (MDPs). In contrast to most decision-theoretic planning frameworks, which assume fully known state transition models, we design a method that eliminates such a strong assumption, which is oftentimes extremely difficult to engineer in reality. To achieve this, we first apply the second-order Taylor expansion of the value function. The Bellman optimality equation is then approximated by a partial differential equation, which only relies on the first and second moments of the transition model. By combining the kernel representation of value function, we then design an efficient policy iteration algorithm whose policy evaluation step can be represented as a linear system of equations characterized by a finite set of supporting states. We have validated the proposed method through extensive simulations in both simplified and realistic planning scenarios, and the experiments show that our proposed approach leads to a much superior performance over several baseline methods.**

I. Introduction

Decision-making of an autonomous mobile robot moving in unstructured environments typically requires the robot to account for uncertain action (motion) outcomes, and at the same time, maximize the long-term return. The Markov Decision Process (MDP) is an extremely useful framework for formulating such decision-theoretic planning problems [7]. Since the robot is moving in a continuous space, directly employing the standard form of MDP needs a discretized representation of the robot state and action. For example, in practice the discretized robot states are associated with spatial tessellation [37], and a grid-map like representation has been widely used for robot planning problems where each grid is regarded as a discrete state; similarly, actions are simplified as transitions to traversable grids which are usually the very few number of adjacent grids in vicinity.

However, the discretization can be problematic. Specifically, if the discretization is low in resolution (i.e., large but few number of grids), the decision policy becomes a very rough approximation of the simplified (discretized) version of the original problem; on the other hand, if the discretization is high in resolution, the result might be approximated well, but this will induce prohibitive computational cost and prevent the real-time decision-making. Finally, the characteristics of state space might be complex and it is inappropriate to conduct lattice-like tessellation which is likely to result in sub-optimal solutions. See Fig. 1 for an illustration.

Fig. 1: In unstructured environments, the robot needs to make motion decisions in the navigable space with spatially varying terrestrial characteristics (hills, ridges, valleys, slopes). This is different from the simplified and structured environments where there are only two types of representations, i.e., either obstacle-occupied or obstacle-free. Evenly tessellating the complex terrain to create a discretized state space cannot effectively characterize the underlying value function used for computing the MDP solution. (Picture credit: NASA)

Another critical issue lies in MDP's transition model which describes the probabilistic transitions from a state to others. However, obtaining an accurate stochastic transition model for robot motion transition is unrealistic even without considering spatiotemporal variability. This is another important factor that significantly limits the applicability of MDP in many real-world problems. Reinforcement learning [19] does not rely on a transition model specification, but requires cumbersome training trials to learn the value function and the policy, which can be viewed as another strong assumption in many robotic missions. Thus, it is desirable that the demanding assumption of a known transition model can be relaxed. Fortunately, the characteristics of transition probabilistic distribution (e.g., mean, variance, or quantiles) for most robotic planning and control systems can be obtained from historical data or offline tests [37]. If we only assume such "partial knowledge"–mean and variance–of the transition model, we must re-design the modeling and solving mechanisms which will be presented in this work.

To address the above problems, we propose a kernel Taylor-based approximation approach. Our contributions can be summarized as follows:

- First, to relax the requirement of fully known transition functions, we apply the second-order Taylor expansion to the value function [8, 9]. The Bellman-type policy evalu-

† Authors contributed equally.

J. Xu and L. Liu are with the Luddy School of Informatics, Computing, and Engineering at Indiana University, Bloomington, IN 47408, USA. E-mail: {xu14, lantao}@iu.edu. K. Yin is with Expedia Group, Inc. E-mail: kyin@expediagroup.com.

ation equation and Bellman optimality equation are then approximated by a partial differential equation (PDE) which only relies on the first and second moments of transition probability distribution.

- Second, to improve the generalizability of the value function, we use kernel functions which can represent a large number of function families for better value approximation. This approximation can conveniently characterize the underlying value functions with a finite set of discrete supporting states.
- Finally, we develop an efficient policy iteration algorithm by integrating the kernel value function representation and the Taylor-based approximation to Bellman optimality equation. The policy evaluation step can be represented as a linear system of equations with characterizing values at the finite supporting states, and the only information needed is the first and second moments of the transition function. This alleviates the need for heavily searching in continuous/large state space and the need for carefully modeling/engineering the transition probability.

II. Related Work

Our work primarily focuses on value function approximation in large/continuous state space using only minimum prior knowledge of the transition function. A major challenge for solving the continuous-state MDP involves the search in a large-scale (usually infinite) state space. Popular methods in robotics that avoid intractability of computing the value function over the continuous space are by tessellating the continuous state space into grids [30, 29, 12, 1, 3]. However, this naive approach does not scale well and may give inferior performance when the problem size increases, known as the curse of dimensionaliy [4]. A more advanced discretization technique that alleviates this problem is by adaptive discretization [15, 23, 38, 27, 22].

Alternative methods tackle this challenge by representing and also approximating the value function by a set of basis functions or some parametric functions [6, 35, 28]. The parameters can be optimized by minimizing the Bellman residual [2]. However, these methods are not applicable in complicated problems because defining features to approximate the value function linearly is non-trivial. This weakness may be resolved through kernel methods [17]. Because the weights in linear combination of basis functions can be presented by their product (through so-called duel form of least squares [33]), the weights can be written in terms of kernel functions and value functions at supporting states. Once value functions at supporting states are obtained, the approximation to value function at any state is also determined. The approach to approximating the value function by kernel functions is referred as the *direct kernel-based method* in this paper. In addition, it is generally hard to find a suitable nonlinear function (such as neural networks) to approximate the value function [13]. There is a vast literature on kernelized value function approximations in reinforcement learning [10, 20, 36, 39], but few studies in robotic planning problems leveraged this approach. A recent application of work [10] on marine robots can be found in [24].

Unfortunately, all these schemes rely on either fully known transitions in MDP or the selection of basis functions, which are difficult to obtain in practice. Therefore, the challenge becomes *how to design a principled methodology without explicitly relying on basis functions and without full knowledge of transitions in MDP, which will be addressed in this work.*

III. Preliminary Material

A. Markov Decision Processes

We formulate the robot decision-theoretic planning problem as an infinite horizon discounted Markov Decision Process (MDP) with continuous states and finite actions. An infinite horizon discounted MDP is defined by a 5-tuple $\mathcal{M} \triangleq \langle \mathbb{S}, \mathbb{A}, T, R, \gamma \rangle$, where $\mathbb{S} = \{s\} \subseteq \mathbb{R}^N$ is the n-dimensional continuous state space and $\mathbb{A} = \{a\}$ is a finite set of actions. $\mathbb{S}$ can be thought of as the robot workspace in our study. A robot transits from a state s to the next s' by taking an action a in a stochastic environment and obtains a reward $R(s, a)$. Such transition is governed by a conditional probability distribution $T(s, a, s') \triangleq p(s'|s, a)$ which is also termed as transition model (or transition function); the reward $R(s, a)$, a mapping from a pair of state and action to a scalar value, specifies the short-term objective that the robot receives by taking action a at state s. The final element $\gamma \in [0, 1)$ in $\mathcal{M}$ is a discount factor which will be used in the expression of value function.

We consider the class of deterministic policies Π, which defines a mapping $\pi \in \Pi : \mathbb{S} \to \mathbb{A}$ from a state to an action. The expected discounted cumulative reward for any policy π starting at any state s is expressed as

$$v^\pi(s) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k)|s_0 = s, a_k = \pi(s_k)], \tag{1}$$

We can rewrite the above equation recursively as follows

$$v^\pi(s) = R(s, \pi(s)) + \gamma \mathbb{E}^\pi[v^\pi(s')|s] \triangleq \mathcal{B}^\pi v^\pi(s), \tag{2}$$

where $\mathcal{B}^\pi$ is called the *Bellman operator*, and $\mathbb{E}^\pi[v^\pi(s')|s] = \int p(s'|s, \pi(s)) v^\pi(s')\, ds'$. The function $v^\pi(s)$ in Eq. (2) is usually called the *state value function* of the policy π. Solving an MDP is to find the optimal policy π^* with the optimal value function which satisfies the Bellman optimality equation

$$v^{\pi^*}(s) = \max_\pi \left\{ R(s, \pi(s)) + \gamma \int p(s'|s, \pi(s)) v^\pi(s')\, ds' \right\}. \tag{3}$$

B. Approximate Policy Iteration via Value Function Representation

To solve an MDP, value iteration and policy iteration are the most prevalent approaches. It has been shown that the value iteration and policy iteration can achieve similar state-of-the-art performances in terms of solution quality and running time [5, 35]. Our work will be built upon policy iteration and here we provide a summary of the important value function approximation process used in the policy iteration [31, 14].

Policy iteration requires initialization of the policy (can be random), based on which a system of linear equations can be established where each equation is exactly the value function (Eq. (2)). When the states in the MDP are finite, the solution to this linear system yields incumbent values for all states [32]. This step is called *policy evaluation*. The second step is to improve the current policy by greedily improving local actions based on the incumbent values obtained. This step is called *policy improvement*. Through iterating these two steps, we can find the optimal policy and a unique solution to the value function that satisfies Eq. (1) for every state.

If, however, the states are continuous or the number of states is infinite, it is difficult to evaluate the value function at every state. One must resort to approximate solutions. Suppose that the value function can be represented by a weighted linear combination of known functions where only weights are to be determined, then a natural way to go is leveraging the Bellman-type equation, i.e., Eq. (2), to compute the weights. Specifically, given an arbitrary policy, the representation of value function can be evaluated at a finite number of states, leading to a linear system of equations whose solutions can be viewed as weights [21]. This obtained representation of value function can be used to improve the current policy. The remaining procedure is then similar to the standard policy iteration method. The final obtained value function representation serves as an approximated optimal value function *for the whole continuous state space*, and the corresponding policy can be obtained accordingly.

Formally, let the value function approximation under policy π be

$$v^{\pi}(s) \simeq v(s; w^{\pi}) = \sum_{i=1}^{m} w_i^{\pi} \cdot \phi_i(s), \tag{4}$$

where $\phi_i \in \Phi \triangleq \{\phi_1, \ldots, \phi_m\}$. The set Φ is the *basis functions* in literature [31]. A finite number of supporting states $\mathbf{s} = \{s^1, \ldots, s^N\}$, $N > m$ can be selected, which are minimized via the squared *Bellman error* over $\mathbf{s}$, defined by $\mathcal{L}(w^{\pi}) = \sum_{i=1}^{N}(v(s^i; w^{\pi}) - \mathcal{B}^{\pi} v(s^i; w^{\pi}))^2$. The solution for w^{π} may have a closed form in terms of the basis functions, transition probabilities, and rewards [21]. By policy iteration, the final solution for $v(s; w^{\pi})$ can be obtained.

Note that $v(s; w^{\pi})$ may be generalized to any parametric nonlinear functions such as neural networks, and that the selection of supporting states $\mathbf{s}$ needs to take account of the characteristics of the underlying value functions (in our robotics decision-theoretic planning scenarios, it relates to the landscape geometry of the terrain).

IV. Kernel Taylor-Based Approximate Policy Iteration

Our objective is to design a principled kernel-based policy iteration approach by leveraging kernel methods to solve the continuous-state MDP. In contrast to most decision-theoretic planning frameworks which assume fully known MDP transition probabilities [7, 32], we propose a method that eliminates such a strong premise which oftentimes is extremely difficult to engineer in practice. To overcome this challenge, first we apply the second-order Taylor expansion of the kernelized value function (Section IV-A). The Bellman optimality equation is then approximated by a partial differential equation which only relies on the first and second moments of transition probabilities (Section IV-B). Combining the kernel representation of value function, this approach efficiently tackles the continuous or large-scale state space search with minimum prerequisite knowledge of state transition model (Sections IV-C and IV-D). Finally, the experiments show that our proposed approach is very powerful and flexible, and reveal great advantages over several baseline approaches (Section V).

A. Taylored Approximate Policy Evaluation Equation

To design an efficient approach for solving MDP based decision-theoretic planning problems, we essentially have two elements to deal with: the value function and the Bellman optimality equation. If we directly apply kernel methods to approximate the value function (referred to as the *direct kernel-based method*), we can avoid explicitly specifying basis functions as mentioned in Section III-B. But it still requires fully known MDP transition probabilities, and it needs the exact Bellman optimality equations to develop the policy iteration method.

In contrast to the direct kernel-based approach, we consider an approximation to the Bellman-type equation by using only first and second moments of transition functions. This will allow us to obtain a nice property that a complete and accurate transition model is not necessary; instead, only the important statistics such as mean and variance (or covariance) will be sufficient. To better describe the basic idea, we keep our discussions on a surface-like terrain and use that surface as the decision-theoretic planning workspace, i.e., $s = [x, y]^T \triangleq [s_x, s_y]^T$, though our approaches apply to the state space of any dimensions.

Formally, suppose that the value function $v^{\pi}(s)$ for any given policy π has continuous first and second order derivatives. We subtract both hand-sides by $v^{\pi}(s)$ from Eq. (2) and then take Taylor expansions of value function around s up to second order [8]:

$$\begin{aligned}
&- R(s, \pi(s)) \\
&= \gamma \left(\mathbb{E}^{\pi}[v^{\pi}(s') \mid s] - v^{\pi}(s) \right) - (1-\gamma)\, v^{\pi}(s) \\
&= \gamma \int p(s'|s, \pi(s))(v^{\pi}(s') - v^{\pi}(s))\, ds' - (1-\gamma)\, v^{\pi}(s) \\
&\simeq \gamma \left((\mu_s^{\pi})^T \nabla v^{\pi}(s) + \frac{1}{2} \nabla \cdot \sigma_s^{\pi} \nabla v^{\pi}(s) \right) - (1-\gamma)\, v^{\pi}(s),
\end{aligned} \tag{5}$$

where μ_s^{π} and σ_s^{π} are the first moment (i.e., mean, a 2-dimensional vector) and the second moment (i.e., covariance, a 2-by-2 matrix) of transition functions, respectively, with the following form

$$(\mu_s^{\pi})_i = \int p(s'|s, \pi(s))(s'_i - s_i)\, ds', \tag{6a}$$

$$(\sigma_s^{\pi})_{i,j} = \int p(s'|s, \pi(s))(s'_i - s_i)(s'_j - s_j)\, ds', \tag{6b}$$

for $i, j \in \{x, y\}$; the operator $\nabla \triangleq (\partial/\partial x, \partial/\partial y)$ and the notation $\cdot$ in the last equation indicate an inner product. To be clear, we present the expression for the following operator

$$\nabla \cdot \sigma_s^\pi \nabla = \sigma_{xx}^\pi \frac{\partial^2}{\partial x^2} + \sigma_{xy}^\pi \frac{\partial^2}{\partial x \partial y} + \sigma_{yx}^\pi \frac{\partial^2}{\partial y \partial x} + \sigma_{yy}^\pi \frac{\partial^2}{\partial y^2}.$$

Since Eq. (5) approximates calculation of Eq. (2) in the policy evaluation stage, the solution to Eq. (5) thus provides the value function approximation under current policy π. Eq. (5) also implies that we only need to use the mean $(\mu_s^\pi)_i$ and convariance $(\sigma_s^\pi)_{i,j}$ instead of the original transition model $p(s'|s, \pi(s))$ to approximate the value function.

B. Approximate Bellman Optimality Equation via PDE

We need to analyze the necessary boundary conditions to Eq. (5) which is a partial differential equation (PDE), and develop an approximation methodology to the Bellman optimality equation which is the foundation for efficient MDP solution. To achieve these, first, the directional derivative of the value function with respect to the unit vector normal at the boundary states must be zero. (Note, the value function should not have values on obstacles or outside the state space $\mathbb{S}$.) Second, in order to ensure a unique solution, we constrain the value function at the goal state to a fixed value.

Let us denote the boundary of entire continuous planning region/workspace by $\partial\mathbb{S}$ and the goal state by s_g. Suppose the value function at s_g is v_g. Section IV-A implies that the Bellman optimality equation Eq. (3) can be approximated by the following PDE:

$$\begin{aligned} 0 \;=\; & \max_\pi \Big\{ \gamma \Big((\mu_s^\pi)^T \nabla v^\pi(s) + \frac{1}{2} \nabla \cdot \sigma_s^\pi \nabla v^\pi(s) \Big) + \\ & R(s, \pi(s)) - (1 - \gamma)\, v^\pi(s) \Big\}, \end{aligned} \tag{7}$$

with boundary conditions

$$\sigma_s^\pi \nabla v^\pi(s) \cdot \hat{\boldsymbol{n}} = 0, \text{ on } \partial\mathbb{S} \tag{8a}$$

$$v^\pi(s_g) = v_g, \tag{8b}$$

where $\hat{\boldsymbol{n}}$ denotes the unit vector normal to $\partial\mathbb{S}$ pointing outward. The condition (8a) is a type of homogeneous Neumann condition, and condition (8b) can be thought of as a Dirichlet condition in literature [11]. This elegantly approximates the classic Bellman optimality equation by a convenient PDE representation. In the next section, we will leverage the kernelized representation of the value function to avoid difficulties of directly solving PDE. The kernel method will help transform the problem to a linear system of equations with unknown values at the finite supporting states.

C. Kernel Taylor-Based Approximate Policy Evaluation

With aforementioned formulations, another critical research question is whether the value function can be represented by some special functions that are able to approximate large function families in a convenient way. We tackle this question by using a kernel method to represent the value function. Thanks to Eq. (7) which allows us to extend with kernelized policy evaluation for Taylored value function approximation.

Specifically, let $k(\cdot, \cdot)$ be a generic kernel function[†] [17]. For a set of selected finite supporting states $\mathbf{s} = \{s^1, \ldots, s^N\}$, let $\mathbf{K}$ be the Gram matrix with $[\mathbf{K}]_{i,j} = k(s^i, s^j)$, and $\mathbf{k}(\cdot, \mathbf{s}) = [k(\cdot, s^1), \ldots, k(\cdot, s^N)]^T$. Given a policy π, assume the value functions at $\mathbf{s}$ are $V^\pi = [v^\pi(s^1), \ldots, v^\pi(s^N)]^T$. Then, for any state s', the kernelized value function has the following form

$$v^\pi(s') = \mathbf{k}(s', \mathbf{s})^T \; (\lambda \mathbf{I} + \mathbf{K})^{-1} \; V^\pi, \tag{9}$$

where $\lambda \geq 0$ is a *regularization factor*. When $\lambda = 0$, it links to the kernel ordinary least squares estimation of w^π in Eq. (4); when $\lambda > 0$, it refers to the ridge-type regularized kernel least squares estimation [33]. Furthermore, Eq. (9) implies that as long as the values V^π are available, the value function for any state can be immediately obtained. Now our objective is to get V^π through Eq. (5) and boundary conditions Eq. (8).

Plugging the kernelized value function representation into Eq. (5), we end up with the following linear system:

$$\Big(\mathbf{M}^\pi \, (\lambda \mathbf{I} + \mathbf{K})^{-1} - (1 - \gamma)\, \mathbf{I} \Big) \; V^\pi = \mathbf{R}^\pi, \tag{10}$$

where $\mathbf{I}$ is an identity matrix, $\mathbf{R}^\pi$ is a $N \times 1$ vector with element $[\mathbf{R}^\pi]_i = -R(s^i, \pi(s^i))$, and $\mathbf{M}^\pi$ is a matrix whose elements are:

$$[\mathbf{M}^\pi]_{i,j} = \gamma \left((\mu_{s^i}^\pi)^T \nabla_{s^i} + \frac{1}{2} \nabla_{s^i} \cdot \sigma_{s^i}^\pi \nabla_{s^i} \right) k(s^i, s^j). \tag{11}$$

Note that ∇_{s^i} indicates the derivatives with respect to s^i, i.e., $\nabla_{s^i} \triangleq (\partial/\partial s_x^i, \partial/\partial s_y^i)$. In Appendix, we provide a concrete example using Gaussian kernels which lead to closed-form expressions and are widely utilized in practice.

The solutions to the system Eq. (10) yield values of V^π. These values further allow us to obtain the value function (9) for any state under current policy π. This completes modeling our kernel Taylor-based approximate policy evaluation framework.

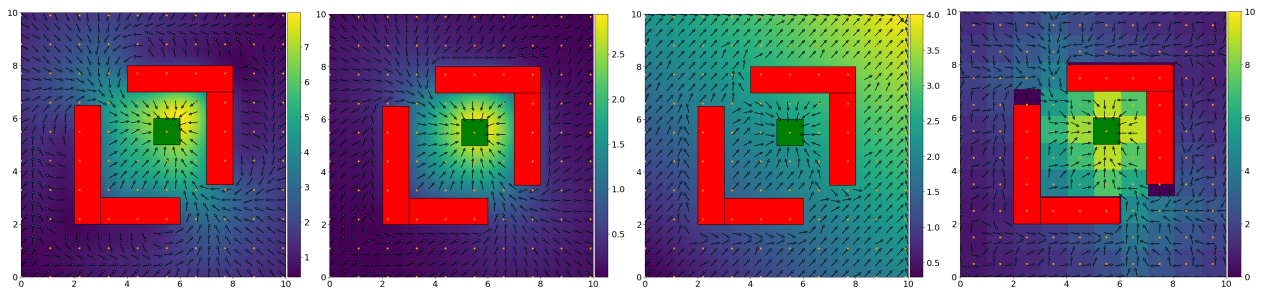
D. Kernel Taylor-Based Approximate Policy Iteration

With the above new model, our next step is to design an implementable algorithm that can solve the continuous-state MDP efficiently. We extend the classic policy iteration mechanism which iterates between the policy evaluation step and the policy improvement step until convergence to find the optimal policy as well as its corresponding optimal value function.

Because our kernelized value function representation depends on the finite supporting states $\mathbf{s}$ instead of the whole state space, we only need to improve the policy on $\mathbf{s}$. Therefore, the policy improvement step in the $(k + 1)$-th iteration is to produce a new policy according to

$$\pi_{k+1}(s) = \arg\max_{a \in \mathbb{A}} \Big\{ R(s, a) + \gamma \Big((\mu_s^a)^T \, \nabla + \frac{1}{2} \nabla \cdot \sigma_s^a \nabla \Big) v^{\pi_k}(s) \Big\} \tag{12}$$

[†] It is worth mentioning that our approach of utilizing kernel methods is to approximate the function. This usage should be distinguished from that in the machine learning literature where kernel methods are used to learn patterns from data.



(a) Kernel Taylor-based PI (b) Direct kernel-based PI (c) NN (d) Grid-based PI

Fig. 2: Evaluation with a traditional simplified scenario where obstacles and goal are depicted as red and green blocks, respectively. We compare the final value function and the final policy obtained from (a) kernel Taylor-based PI, (b) direct kernel-based PI, (c) NN, and (d) grid-based PI. A brighter background color represents a higher state value. The policies are the arrows (vector fields), and each arrow points to some next waypoint. Orange dots denote supporting states or the grid centers (in the case of grid-based PI).

Algorithm 1 Kernel Taylor-Based Approximate Policy Iteration

```
Input: A set of supporting states $\mathbf{s} = \{\mathbf{s}^1, ..., \mathbf{s}^N\}$; the kernel
    function $k(\cdot,\cdot)$; the regularization factor $\lambda$; the MDP
    $\langle \mathbb{S}, \mathbb{A}, T, R, \gamma \rangle$.
Output: The kernelized value function Eq. (9) for every state
    and corresponding policy.
 1: Initialize the action at the supporting states.
 2: Compute the matrix $\mathbf{K} + \lambda \mathbf{I}$ and its inverse.
 3: repeat
 4:     // Policy evaluation step
 5:     Solve for $V^\pi$ according to Eq. (10) in Section IV-A.
 6:     // Policy improvement step
 7:     for $i = 1, ..., N$ do
 8:         Update the action at the supporting state $s^i$ based
    on Eq. (12).
 9:     end for
10: until actions at the supporting states do not change.
```

where $s \in \mathbf{s}$, π_k and π_{k+1} are the current policy and the updated policy, respectively. Note that μ_s^a and σ_s^a depend on a through the transition function $p(s'|s, a)$ in Eq. (6). Compared with the approximated Bellman optimality equation (Eq. (7)), Eq. (12) drops the term $(1-\gamma)v^{\pi_k}(s)$. This is because $v^{\pi_k}(s)$ does not explicitly depend on action a. The value function of the updated policy satisfies $v^{\pi_{k+1}}(s) \geq v^{\pi_k}(s)$ [5]. If the equality holds, the iteration converges.

The final kernel Taylor-based policy iteration algorithm is pseudo-coded in Alg. 1. It first initializes the actions at the finite supporting states and then iterates between policy evaluation and policy improvement. Since the supporting states as well as the kernel parameters do not change, the regularized kernel matrix and its inverse are computed only once at the beginning of the algorithm. This greatly reduces the computational burden caused by matrix inversion. Furthermore, due to the finiteness of the supporting states, the entire algorithm views the policy π as a table and only updates the actions at the supporting states using Eq. (12). The algorithm stops and returns the supporting state values when the actions are stabilized. We can then use these state-values to get the final kernel value function that approximates the optimal solution. The corresponding policy for every continuous state can then be easily obtained from this kernel value function [34].

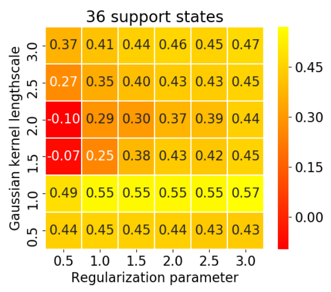
Intuitively, this proposed framework is flexible and powerful due to the following reasons: rather than tackling the difficulties in solving the PDE which approximates the original Bellman-type equation, we use the kernel representation to convert the problem to a system of linear equations with characterizing values at the finite discrete supporting states. From this viewpoint, our proposed method nicely balances the trade-off between searching in finite states and that in infinite states. In other words, our approach leverages the kernel methods and Bellman optimal conditions under the practical assumptions.

V. Experiments

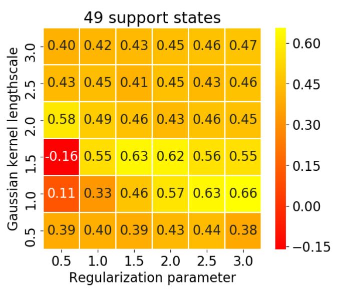
To validate our method, we consider two mobile robot decision-theoretic planning tasks. The first one is a goal-oriented planning problem in a simple environment with obstacle-occupied and obstacle-free spaces. This will help us to evaluate basic algorithmic properties. In the second task, we demonstrate that our method can be applied to a more realistic as well as more challenging navigation scenario on Mars surface [25], where the robot needs to take the elevation of the terrain surface into account (i.e., "obstacles" are implicit). In both tasks, we assume that the estimates of the first two moments of the transition probability are obtained from the past field experiments. To be concrete, in both experiments we use the Gaussian kernel given in Appendix A.

A. Plane Navigation

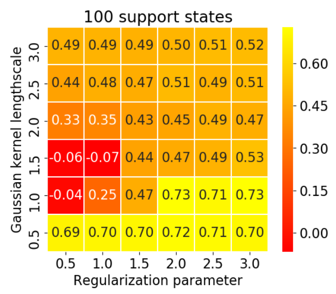
1) Setup: Our first experiment is a 2D plane navigation problem, where the obstacles and a goal area are represented in a $10m \times 10m$ environment, as shown in Fig. 2. The state space for this task is a 2-dimensional euclidean space, i.e., $s = [s_x, s_y]^T$ and $s \in \mathbb{S} \subseteq \mathbb{R}^2$. The action space is a finite set of Q points $\mathbb{A}(s) = \{a_i(s) | i \in \{1, ..., Q\}\}$. Each point $a_i(s) = [s_x + r\cos(\frac{2\pi i}{Q}), s_y + r\sin(\frac{2\pi i}{Q})]^T$ is an action generated on



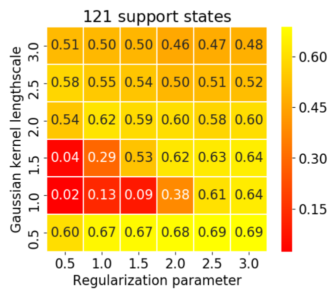
(a) 6×6



(b) 7×7



(c) 10×10



(d) 11×11

Fig. 3: The performance matrix obtained by the hyperparameter search using (a) 6×6; (b) 7×7; (c) 10×10; and (d) 11×11 evenly-spaced supporting states. Rows and columns represent different Gaussian kernel lengthscale and regularization parameters, respectively. The numbers in the heatmap represent the average return of the final policy obtained using the corresponding hyperparameter combination. The colorbar is shown on the right side of each table.

a circle centered at the current state with a radius r. In this experiment, we set the number of actions $Q = 12$ and the action radius as $r = 0.5m$. An action point can be viewed as the "carrot-dangling" waypoint for the robot to follow, which serves as the input to the low-level motion controller. For the reward function, we set the reward of arriving at the goal and obstacle states to be $+1$ and -1, respectively. Since the reward now depends on the next state, we use Monte-carlo sampling to estimate the expectation of $R(s, a)$. The discount factor for the reward is set to $\gamma = 0.9$. We set the obstacle areas and the goal as absorbing states, i.e., the robot can not transit to any other states if they are in these states. To satisfy the boundary condition mentioned in Section IV-B, we allow the robot to receive rewards at the goal state, but it cannot receive any reward if its current state is within an obstacle. Thus, the goal state value is $\frac{1}{1-\gamma} = 10$.

2) *Performance measure:* Since the ultimate goal of planning is to find the optimal policy, our performance measure is based on the quality of the policy. A policy is better if it achieves a higher expected cumulative reward starting from every state. Because it is impossible to evaluate over the infinite number of states, we numerically evaluate the quality of a policy using the average return criterion [18]. In detail, we first uniformly sample $Q = 10^4$ states to ensure a thorough performance evaluation. Then, for each sampled state, we execute the policy to generate multiple trajectories, where each

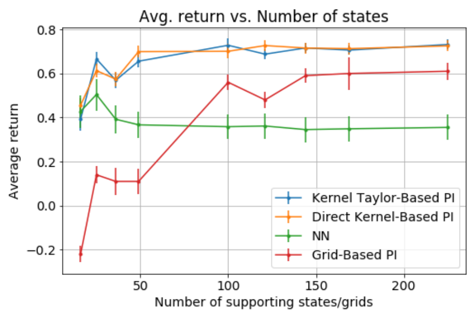


Fig. 4: The comparison of the average return of the policies computed from the four algorithms. The x-axis is the number of supporting states/grids used in computing the policy. The y-axis shows the average return.

trajectory ends when it arrives at a terminal state (goal or obstacle) or reaches an allowable maximal number of steps. This procedure gives us an expected performance of the policy at any state by averaging the discounted sum of rewards over all the trajectories starting from it. Now, we can calculate the average return criterion by averaging over the performance of sampled states. A higher value of the average return implies that, on average, the policy gives better performance over the entire state space.

3) *Results:* To evaluate the effect of supporting states, we place evenly-spaced supporting states (in a lattice pattern) with different spacing resolution. Besides the number of states, the kernel lengthscale (see Appendix A) and the regularization parameter λ are the other two *hyperparameters* governing the performance of our algorithm. We present the grid-based hyperparameter search results using four different configurations of supporting states shown in Fig. 3. The lengthscale and regularization parameters are searched over the same values, $\{0.5, 1, 1.5, 2, 2.5, 3\}$. By entry-wise comparison among the four matrices in Fig. 3, we can observe that increasing the number of states leads to improving performance in general. However, we can find that the best performed policy is given by the 10×10 supporting states configuration (Fig. 3(c)) which is not the scenario with the best spacing resolution. This indicates that *a larger number of states can also result in a deteriorating solution*, and the performance of the algorithm is a matter of *how the supporting states are placed (distributed)*, instead of *the number (resolution) of state discretization*. Furthermore, we can gain some insights on how to select the hyperparameters based on the number of supporting states. Low-performing entries (highlighted with red) occur more often on the left side of the performance matrix when the number of supporting states increases. It implies that with more supporting states, the algorithm requires a stronger regularization (i.e., greater λ described in Section IV-C). On the other hand, high-performing policies (indicated by yellow) appear more on the bottom of the performance matrix when a greater number of supporting states present, which means that a smaller length scale is generally required given a larger quantity of supporting states.

We further compare our kernelized value function repre-

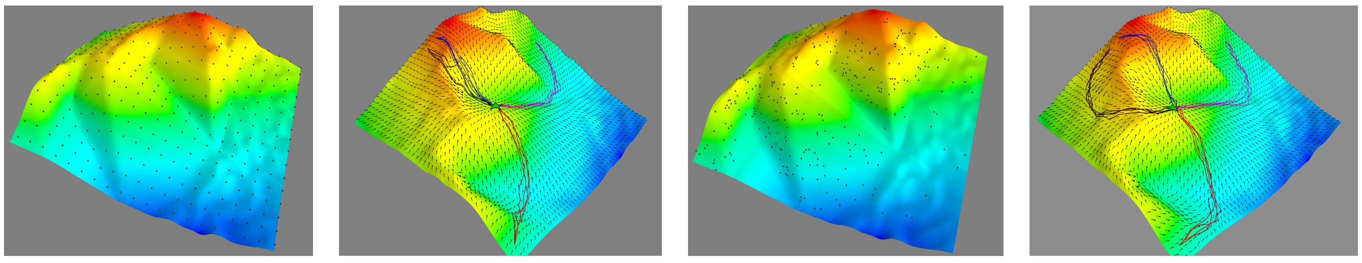


Fig. 5: Supporting state distribution and the policy for evenly-spaced selection and importance sampling-based selection. The 3D surface shows the Mars digital terrain model obtained from HiRISE. Supporting states and the policies are shown in black dots and vector fields, respectively. The colored lines represent the sampled trajectories, which initiate from four different starting positions. (a)(b) The evenly-spaced supporting states and the corresponding policy and trajectories; (c)(d) The supporting states generated by importance sampling and the corresponding policy and trajectories.

sentation against other three variants of the value function approximations. Specifically, the first one is the direct kernel-based approximation method using a Gaussian kernel. This method is similar to the one in [20], but with a fully known transition function. The second one uses neural networks (NNs) as the value function approximator. We setup the NN configuration similar to a recent work [16]. In detail, we use a shallow two-layer network with 100 hidden units in each layer. Its parameters are optimized through minimizing the squared Bellman error via gradient descent. Since there is no closed-form solution to compute the expected next state value when using NN as a function approximator, we use Monte-Carlo sampling to estimate the expected value at the next states $\mathbb{E}[v^{\pi}(s')|s]$. The third method is the grid-based approximation method. It first transforms the continuous MDP to a discrete version, where each state is regarded as a grid. Then, it uses the vanilla policy iteration to solve the discretized MDP.

The comparison among above methods aims at investigating two important questions:

1) How does the kernelized value function representation compare to other representations (NN and grid-based) in terms of the final policy performance?
2) Compared to our method, the direct kernel-based method not only requires the fully known transition function, but also restricts the transition to be a Gaussian distribution. Can our method with only mean and variance obtain similar performance as the direct kernel-based method?

To answer these questions, we choose the transition function as a Gaussian distribution. Its mean is the selected next waypoint. We set the standard deviation of the transition function to be $0.2m$ on both axes during the experiment. The transition probability models the accuracy of the low-level motion controller: more accurate controller leads to smaller uncertainty. To perform fair comparisons, we use the same supporting states and apply hyperparameter search to all the methods.

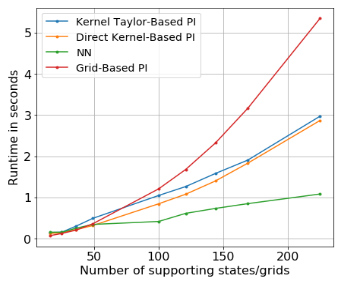
The results for four methods are shown in Fig. 4 in terms of the average return. The first question is answered by the fact that the kernel-based methods (kernel Taylor-based and direct kernel-based PI) consistently outperforms the other two methods. Moreover, our method has the performance as good as the direct kernel-based method which however requires the prerequisite full distribution information of the transition. This indicates that our method can be applied to broader applications that do not have full knowledge of transition functions. In contrast to the grid-based PI, the kernel-based algorithms and NN can achieve moderate performance even with a small number of supporting states. It indicates that the continuous representation of the value function is crucial when supporting states are sparse. However, increasing the number of states does not improve the performance of the NN.

In Fig. 6, we compare the computational time and the number of iterations to convergence. The computational time of our method is less than the grid-based method as revealed in Fig. 6(a). We notice that there is a negligible computational time difference between our method and the direct method. As a parametric method, NN has the least computational time, and only linearly increases, but it does not converge as indicated by Fig. 6(b).

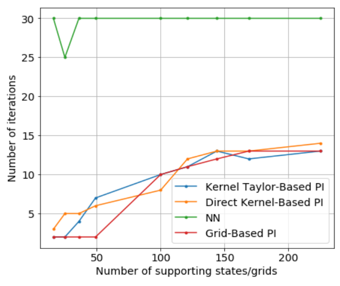
The function values and the final policies are visualized in Fig. 2. All the methods except for the NN obtain reasonable approximations to the optimal value function. Compared to our method, the values generated by the grid-based method are discrete "color blocks", thus the obtained policy is non-smooth. The direct kernel-based method obtains a slightly more "aggressive" (dangerous) policy in contrast to our method.

B. Martian Terrain Navigation

In this experiment, we consider the autonomous navigation task on the surface of Mars with a rover. We obtain the Mars terrain data from *High Resolution Imaging Science Experiment* (HiRISE) [26]. Since there is no explicitly presented "obstacle", the robot only receives the reward when it reaches the goal. If the rover attempts to move on a steep slope, it may be damaged and trapped within the same state with probability proportional to the slope angle. Otherwise, its next state is distributed around the desired waypoint specified by the current action. This indicates that the underlying transition function should be the mixture of these two situations. It is reasonable to assume that the means of the two cases are given

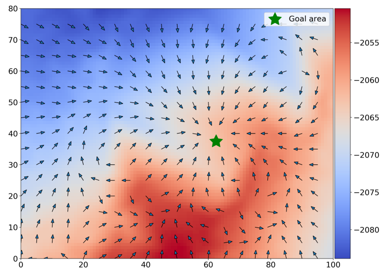


(a)

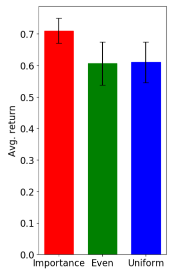


(b)

Fig. 6: Computational time comparisons of the four algorithms with changing number of states. (a) The computational time per iteration. (b) Number of iterations to convergence.



(a)



(b)

Fig. 7: (a) The top-down view of the Mars terrain surface as well as the policy generated by our method with the importance sampling selection. The colormap indicates the height (in meters) of the terrain. (b) The comparison of average return among three supporting state selection methods using the same number of states. Red, green, and blue bars indicate the performance of importance sampling selection, evenly-spaced selection, and uniform distribution sampling selection, respectively.

by the current state and the next waypoint, respectively. We can similarly have an estimate of the variances. The mean and the variance of the transition function can then be computed using the law of total expectation and total variance, respectively.

Due to the complex and unstructured terrestrial features, evenly-spaced supporting state points may fail to best characterize the underlying value function. Also, to keep the computational time at a reasonable amount while maintaining a good performance, we leverage the importance sampling technique to sample the supporting states that concentrate around the dangerous regions where there are steep slopes. This is obtained by first drawing a large number of states uniformly covering the whole workspace. For each sampled state, we then assign a weight proportional to its slope angle. Finally, we resample supporting states based on the weights. To guarantee the goal state to have a value, we always place one supporting state at the center of the goal area.

Fig. 5(a) and Fig. 5(c) compare the two methods for supporting state selections. The supporting states given by the importance sampling-based method are dense around the slopes. These supporting states better characterize the potentially high-cost and dangerous areas than the evenly-spaced selection scheme. We selected four starting locations from where the rover needs to plan paths to arrive at a goal location. For each starting location, we conducted multiple trials following the produced optimal policies. The trajectories generated with the importance sampling states in Fig. 5(d) attempt to approach the goal (green star) with minimum distances, and at the same time, avoid high-elevation terrains. In contrast, the trajectories obtained using the evenly-spacing states in Fig. 5(b) approach the goal in a more aggressive manner which can be risky in terms of safety. It indicates that a good selection of supporting states can better capture the state value function and thus produce finer solutions. This superior performance can also be reflected in Fig. 7(b). The policy obtained by the uniformly sampled states shows similar performance to the one generated by the evenly-spacing states, both of which yield smaller average return than the importance-sampled case. A top-down view of the policy is shown in Fig. 7(a) where the background colormap denotes the elevation of terrain.

VI. Conclusion

This paper presents an efficient policy iteration algorithm to solve the continuous-state Markov Decision Process by integrating the kernel value function representation and the Taylor-based approximation to Bellman optimality equation. Our algorithm alleviates the need for heavily searching in continuous state space and the need for precisely modeling the state transition functions. We have thoroughly evaluated the proposed method through simulations in both simplified and realistic planning scenarios. The experiments comparing with other baseline approaches show that our proposed framework is powerful and flexible, and the performance statistics reveal superior efficiency and accuracy of our algorithms.

Appendix

A. Gaussian Kernel for kernel Taylor-Based Approximate Policy Evaluation

Because Gaussian kernels are widely used in the studies of kernel methods, this section presents the necessary derivations to aid the application of Gaussian kernels to our proposed kernel Taylor-Based approximate methods.

Gaussian kernel functions on states s' and s have the form $k(s',s) = c \times \exp\left((-\frac{1}{2}(s'-s)^T\Sigma_s^{-1}(s'-s)\right)$, where c is a constant and Σ_s is a covariance matrix. Note that Σ_s is referred to as the length-scale parameter in our paper. Due to limited space, we only provide formula below for the first and second derivatives of the Gaussian kernel functions. These formula are necessary when Gaussian kernels are employed (for example, Eq.(11)). In fact, we have

$$\nabla_{s'} k(s',s) = -\Sigma_1^{-1}(s'-s)k(s',s), \tag{13}$$

and

$$\begin{aligned}\nabla_{s'} \cdot \sigma_s \nabla_{s'} k(s',s) &= -tr(\sigma_s\,\Sigma_s^{-1})\,k(s',s)\\ &+ (s'-s)^T\Sigma_s^{-T}\sigma_s\,\Sigma_s^{-1}(s'-s)k(s',s),\end{aligned} \tag{14}$$

where $tr(\cdot)$ denotes the trace of the matrix.

References

[1] Wesam H Al-Sabban, Luis F Gonzalez, and Ryan N Smith. Wind-energy based path planning for unmanned aerial vehicles using markov decision processes. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 784–789. IEEE, 2013.

[2] András Antos, Csaba Szepesvári, and Rémi Munos. Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129, 2008.

[3] Stanley S Baek, Hyukseong Kwon, Josiah A Yoder, and Daniel Pack. Optimal path planning of a target-following fixed-wing uav using sequential decision processes. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2955–2962. IEEE, 2013.

[4] Richard E Bellman. *Adaptive control processes: a guided tour*, volume 2045. Princeton university press, 2015.

[5] Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

[6] Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*, volume 5. Athena Scientific Belmont, MA, 1996.

[7] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[8] Anton Braverman, Itai Gurvich, and Junfei Huang. On the taylor expansion of value functions. *arXiv preprint arXiv:1804.05011*, 2018.

[9] Jonas Buchli, Farbod Farshidian, Alexander Winkler, Timothy Sandy, and Markus Giftthaler. Hamilton-Jacobi-Bellman Equation. In *Optimal and Learning Control for Autonomous Robots*, chapter 1.3. arXiv preprint arXiv:1708.09342, 2017.

[10] Yaakov Engel, Shie Mannor, and Ron Meir. Bayes meets bellman: The gaussian process approach to temporal difference learning. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 154–161, 2003.

[11] Lawrence C. Evans. *Partial Differential Equations: Second Edition (Graduate Series in Mathematics)*. American Mathematical Society, 2010.

[12] Yu Fu, Xiang Yu, and Youmin Zhang. Sense and collision avoidance of unmanned aerial vehicles using markov decision process and flatness approach. In *2015 IEEE International Conference on Information and Automation*, pages 714–719. IEEE, 2015.

[13] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[14] Geoffrey J Gordon. Approximate solutions to markov decision processes. Technical report, Carnegie-Mellon University School of Computer Science, 1999.

[15] Alex A Gorodetsky, Sertac Karaman, and Youssef M Marzouk. Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition. In *Robotics: Science and Systems*, 2015.

[16] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

[17] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008.

[18] Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

[19] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[20] Malte Kuss and Carl E Rasmussen. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 751–758, 2004.

[21] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149, 2003.

[22] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[23] Lantao Liu and Gaurav S Sukhatme. A solution to time-varying markov decision processes. *IEEE Robotics and Automation Letters*, 3(3):1631–1638, 2018.

[24] John Martin, Jinkun Wang, and Brendan Englot. Sparse gaussian process temporal difference learning for marine robot navigation. *arXiv preprint arXiv:1810.01217*, 2018.

[25] Michel Maurette. Mars rover autonomous navigation. *Autonomous Robots*, 14(2-3):199–208, 2003.

[26] Alfred S McEwen, Eric M Eliason, James W Bergstrom, Nathan T Bridges, Candice J Hansen, W Alan Delamere, John A Grant, Virginia C Gulick, Kenneth E Herkenhoff, Laszlo Keszthelyi, et al. Mars reconnaissance orbiter's high resolution imaging science experiment (hirise). *Journal of Geophysical Research: Planets*, 112 (E5), 2007.

[27] Rémi Munos and Andrew Moore. Variable resolution discretization in optimal control. *Machine learning*, 49 (2-3):291–323, 2002.

[28] Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(May):815–857, 2008.

[29] Michael Otte, William Silva, and Eric Frew. Any-time path-planning: Time-varying wind field+ moving obstacles. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2575–2582. IEEE, 2016.

[30] Arvind A Pereira, Jonathan Binney, Geoffrey A Hollinger, and Gaurav S Sukhatme. Risk-aware path planning for autonomous underwater vehicles using predictive ocean models. *Journal of Field Robotics*, 30(5):

741–762, 2013.

[31] Warren B Powell. Perspectives of approximate dynamic programming. *Annals of Operations Research*, 241(1-2): 319–356, 2016.

[32] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[33] John Shawe-Taylor, Nello Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[34] Jennie Si, Andrew G Barto, Warren B Powell, and Don Wunsch. *Handbook of learning and approximate dynamic programming*, volume 2. John Wiley & Sons, 2004.

[35] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[36] Gavin Taylor and Ronald Parr. Kernelized value function approximation for reinforcement learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1017–1024, 2009.

[37] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*, volume 1. MIT press Cambridge, 2000.

[38] Junhong Xu, Kai Yin, and Lantao Liu. Reachable space characterization of markov decision processes with time variability. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019. doi: 10.15607/RSS.2019.XV.069.

[39] Xin Xu, Dewen Hu, and Xicheng Lu. Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4):973–992, 2007.