Article

Kernel-based diffusion approximated Markov decision processes for autonomous navigation and control on unstructured terrains

The International Journal of Robotics Research
2024, Vol. 43(7) 1056–1080
© The Author(s) 2024
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/02783649231225977
journals.sagepub.com/home/ijr
Sage

Junhong Xu[1], Kai Yin[2], Zheng Chen[1], Jason M Gregory[3], Ethan A Stump[3] and Lantao Liu[1]

Abstract

We propose a diffusion approximation method to the continuous-state Markov decision processes that can be utilized to address autonomous navigation and control in unstructured off-road environments. In contrast to most decision-theoretic planning frameworks that assume fully known state transition models, we design a method that eliminates such a strong assumption that is often extremely difficult to engineer in reality. We first take 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 the value function, we 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 first validate the proposed method through extensive simulations in 2D obstacle avoidance and 2.5D terrain navigation problems. The results show that the proposed approach leads to a much superior performance over several baselines. We then develop a system that integrates our decision-making framework with onboard perception and conduct real-world experiments in both cluttered indoor and unstructured outdoor environments. The results from the physical systems further demonstrate the applicability of our method in challenging real-world environments.

Keywords

Motion planning and control, off-road navigation, Markov decision processes, diffusion approximation, second-order HJB equation, kernel methods

Received 12 March 2023; Revised 28 November 2023; Accepted 19 December 2023
Senior Editor: Tim Barfoot
Associate Editor: Joshua Marshall

1. Introduction

The 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 a very useful framework for formulating such decision-theoretic planning problems (Boutilier et al., 1999). Since the robot is moving in a continuous space, directly employing the standard form of MDP needs a discretized representation of the robot's state and action. For example, in practice, the discretized robot states are associated with spatial tessellation (Thrun et al., 2000), 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 numbers of adjacent grids in the vicinity.

However, discretization can be problematic. Specifically, if the discretization is low in resolution (i.e., large but few numbers 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 costs and prevent 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 Figure 1 for an illustration.

Another critical issue lies in MDP's transition model, which describes the probabilistic transitions between states. However, obtaining an accurate probability distribution function for robot motion transitions is oftentimes unrealistic,

[1]Indiana University-Bloomington, Bloomington, IN, USA
[2]Expedia Group, Austin, TX, USA
[3]Army Research Laboratory, Adelphi, MD, USA

Corresponding author:
Lantao Liu, Luddy School of Informatics, Computing, and Engineering, Indiana University-Bloomington, 2401 N Milo B Sampson Ln, Room 080, Bloomington, IN 47408, USA.
Email: lantao@iu.edu

Figure 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, that is, 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).

even without considering spatiotemporal variability. This is another important factor that significantly limits the applicability of MDP in many real-world problems. Reinforcement learning (Kober et al., 2013) does not rely on a transition model specification but requires a large number of training trials to learn the value function and the policy, which can be viewed as another strong and difficult assumption in many robotic missions. Thus, it is desirable that the demanding assumption of the known transition probability function can be relaxed as some non-exact form and can be obtained without cumbersome learning trials. This can be achieved by leveraging the major characteristics of the transition probabilistic distribution (e.g., moments or quantile of the distribution), which typically can be obtained (well approximated) from lightweight historical data or offline tests (Thrun et al., 2000). If we only assume such "partial knowledge"—first two moments—of the transition model, we must re-design the modeling and solving mechanisms.

To address the above problems, we propose a diffusion-approximated and kernel-based solution. Our primary focus is to develop a theoretically-sound and practically-efficient solution to compute the optimal value function in continuous-state MDPs without requiring a full probabilistic transition model.

Our contributions can be summarized as follows:

- First, we apply the second-order Taylor expansion to the value function to relax the requirement of fully known transition functions. The Bellman-type policy evaluation equation and the Bellman optimality equation are then approximated by a diffusion-type partial differential equation (PDE), which only relies on the first and second moments of the transition probability distribution. The solution of the Taylored Bellman optimality equation can be viewed as an infinite-horizon discounted reward diffusion process.
- Second, to improve both the efficiency and the flexibility of the value function approximation, 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, leading to fast computation.
- Finally, we develop an efficient policy iteration algorithm by integrating the kernel value function representation and the Taylor-based diffusion-type 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.

The organization of the paper is as follows: we review the relevant literature and provide background in formulating stochastic planning problems and value function approximation methods in Section 2 and 3, respectively. In Section 4, we present the Taylored Bellman equation and the proposed policy iteration algorithm. Section 5 provides evaluation results for various important algorithmic properties of the proposed method. Then, we integrate the perception into our framework to build an autonomy system and demonstrate real-world experiments in Sections 6–8. Finally, we conclude the paper and discuss insights learned from the real-world experiments in Section 9. This paper extends our prior conference publication (Xu et al., 2020) by providing technical details to justify the methods within Section 4, integrating up-to-date literature, and conducting extensive experiments, particularly in real-world settings delineated in Sections 6–8.

2. Related work

Our work closely relates to value-based methods for solving MDPs and stochastic path and motion planning in robotics. In this section, we provide concise surveys of the two fields respectively and highlight our contributions at last.

2.1. Value-based methods for solving continuous-state MDPs

Motion planning under action uncertainty can be framed as an MDP with continuous state and action (Bertsekas 2012; Sutton and Barto 2018). Many existing works rely on value-based methods for solving MDPs. This approach focuses on computing a value function from which the policy is derived implicitly using the Bellman optimality equation. However, the intractability of an exact value function representation emerges when dealing with a large or continuous state space, as it requires enumerating over an infinite-sized table that stores every state value. Thus, a large body of research focuses on value function approximation techniques (Kober et al., 2013; Sutton and Barto 2018).

One simple approach to approximating the value function is tessellating the continuous state space into finite uniform grids. This method is popular in many robotic applications (Al-Sabban et al., 2013; Baek et al., 2013; Fu et al., 2015; Otte et al., 2016; Pereira et al., 2013). However, this uniform tessellation approach does not scale well to environments with complex geometric structures, and it requires a large number of grids when the problem size increases, known as the curse of dimensionality (Bellman 2015). A more advanced discretization technique that alleviates this problem is by adaptive discretization (Gorodetsky et al., 2015; LaValle 2006; Liu and Sukhatme 2018; Munos and Moore 2002; Xu et al., 2019).

Alternative methods tackle this challenge by representing and also approximating the value function by a linear combination of basis functions or some parametric functions (Bertsekas and Tsitsiklis 1996; Munos and Szepesvári 2008; Sutton and Barto 2018). The weights in the linear combination can be optimized by minimizing the Bellman residual (Antos et al., 2008). However, these methods do not apply to complicated problems because selecting a proper set of basis functions is non-trivial. This weakness may be resolved through kernel methods (Hofmann et al., 2008). Because the weights in a linear combination of basis functions can be presented by their product (through the so-called duel form of least squares (Shawe-Taylor et al., 2004), the product may be better replaced by kernel functions (on so-called reproducing kernel Hilbert spaces) at supporting states. Once value functions at supporting states are obtained, the approximation to the value function at any state is also determined. In Deisenroth et al. (2009), the authors use a Gaussian process with a Radial Basis Kernel (RBF) to approximate the value function. This approach to approximating the value function by kernel functions is referred as the *direct kernel-based method* in this paper. There is a vast literature on kernelized value function approximations in reinforcement learning (Engel et al., 2003; Kuss and Rasmussen 2004; Taylor and Parr 2009; Xu et al., 2007). But few studies in robotic planning problems leveraged this approach. A recent application of work (Engel et al., 2003) on marine robots can be found in Martin et al. (2018).

More recently, due to the advancement of deep learning, neural networks are proposed to approximate value functions in high-dimensional state spaces (Arulkumaran et al., 2017) such as images (Nair et al., 2018b) and high-dimensional control problems (Lillicrap et al., 2015). Additionally, many state-of-the-art reinforcement learning (RL) algorithms integrate value-based and policy gradient methods, known as actor-critic (Schulman et al., 2017). Unlike the value-based method, whose policy is derived implicitly, the actor-critic framework learns a policy directly using the policy gradient computed from the value function approximated by a neural network. Through the integration of physics simulation, the techniques within this framework have demonstrated successful real-world applications, particularly in controlling locomotion across unstructured terrains (Das et al., 2022; Makoviychuk et al., 2021; Miki et al., 2022; Wang et al., 2021). These methods rely on sampling to compute the value function update via the Bellman equation. It can be sample-inefficient because numerous next-state samples are typically required to compute a Bellman update. In contrast, our proposed method increases the sample efficiency by converting the Bellman equation to a PDE. This operation eliminates the need for sampling next states, requiring only the evaluation of partial derivatives for Bellman updates, bypassing the potentially expensive sampling process. We demonstrate this property in Section 5.

Furthermore, all the above-mentioned existing approaches rely on fully known transitions in MDP, require samples from the complex stochastic motion model, or rely on careful selection of basis functions, which are difficult to design 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.*

2.2. Deterministic and stochastic planning in robotics

Most of the traditional robotic planning methods (see Gammell and Strub (2021) for a recent survey) use a simplified deterministic robot motion model, for example, holonomic kinematics, to compute an open-loop path, for example, sampling-based motion planners such as probabilistic roadmap (PRM) and rapidly exploring random tree (RRT) and their variants (Karaman and Frazzoli 2011; LaValle 2006). Since the modeling error exists between the model used in planning and the real world, the robot cannot execute the path directly and requires a separate controller to track the path. To ensure the path is trackable by the controller, refining the planned path into a kinodynamically trackable trajectory (Webb and Berg 2012) is necessary. One strategy to generate such trajectories is based on trajectory optimization. Existing trajectory optimization methods can be categorized into hard-constrained methods and soft-constrained methods. Using hard-constrained optimization technique to solve trajectory generation problem has been proposed by Mellinger and Kumar (2011), where piecewise polynomial trajectories are generated by solving a quadratic programming (QP) problem. A closed-form solution is provided to solve an unconstrained QP problem, and intermediate waypoints are added iteratively to ensure the safety of the trajectories (Richter et al., 2016). Free space represented by multiple geometric volumes, for example, cubes (Chen et al., 2015; Gao et al., 2018), spheres (Gao et al., 2019; Gao and Shen 2016), and polyhedra (Deits and Tedrake 2015; Liu et al., 2017), are used to formulate a convex optimization problem, which generates smooth trajectories within the volumes. However, hard-constrained methods ignore the distance information to the obstacle boundaries and are prone to generate trajectories close to the obstacles. This can cause collisions when the robot motion model is imperfect. Such safety issue motivates the development of the soft-constrained methods which leverage the distance gradient during optimization such that the trajectories could be adjusted to stay far from obstacle boundaries while maintaining the smoothness property (Oleynikova et al., 2016; Zhou et al. 2019, 2020).

The above design of the navigation system is based on deterministic models at all levels, that is, from path planning to trajectory optimization, and uncertainties in the motion are entirely handled by the tracking controller. Such decoupled design is brittle because it is possible that the feedback controller cannot stabilize on the open-loop path computed by the planner. Therefore, to reduce the burden of designing and tuning separate controllers for different environments, it is desirable to develop methods that directly consider the action execution uncertainty during the planning phase if the error between the planning model and the real-world action is large. This problem is particularly obvious when the robot moves on rough terrains.

Early work of stochastic planning in the robotics community was mostly built upon the sampling-based motion planning paradigm. In Tedrake et al. (2010), the authors compute local feedback controllers while adding new tree branches in the RRT algorithm to stabilize the robot's motion between two tree nodes. A similar idea has also been exploited by Agha-Mohammadi et al. (2014) with a different planner and controller. This method considers both the sensing and motion uncertainty and uses the linear-quadratic Gaussian (LQG) controller (Kalman et al., 1960) to stabilize the motion and sensing uncertainties along the edges generated by a PRM algorithm. Alternatively, Van Den Berg et al. (2011) use the LQG controller to compute a distribution of paths, from which the path is planned by the RRT algorithm that optimizes an objective function based on the path distribution. Instead of modular design of the planning and control components, Huynh et al. (2016) propose a more integrated approach to solving the general continuous-time stochastic optimal control problem. The method first approximates the original MDP by a discrete MDP through sampling points in the state space, and then a closed-loop policy is computed in this discrete MDP. The approximation and solution are then refined by iterating this sampling and solving procedure.

Model predictive control (MPC), or receding horizon control, is another strategy to resolve motion uncertainty (Bertsekas 2005; Rawlings et al., 2017). At each timestep, it solves a finite-horizon optimal control problem and applies the first action of the computed open-loop trajectory online. To reduce computational burden, MPC usually uses a deterministic model to predict future states (Bertsekas 2012). Robust MPC (Bemporad and Morari 1999) explicitly deals with the model uncertainty by optimizing over feedback policies rather than open-loop trajectories. However, this optimization is computationally expensive. Thus, in practice, tube MPC that focuses only on a sub-space is often used as an approximate solution to robust MPC (Langson et al., 2004). Tube MPC for nonlinear robot motion models is still an active research area. For example, sum-of-squared programming (Majumdar and Tedrake 2013, 2017) and reachability analysis (Althoff et al., 2008; Bansal et al., 2017) are used for solving the nonlinear tube MPC problem.

In general, robot motion planning essentially requires to reason about uncertainties during motion control. However, because the prohibitive computation prevents the robot from real-time decision, most of the traditional approaches to generating feasible trajectories do not consider uncertainties. Oftentimes some heuristics are leveraged to ensure the motion safety during execution, for example, using the soft-constrained methods. In contrast, our proposed framework solves the stochastic planning problem in a direct and integrative fashion. As a result, the computed policy can naturally guide the robot with a large clearance from the obstacles, without adding extra penalty in the objective function which can be hard to specify in practice. In addition, most existing methods that solve the stochastic planning problems avoid the difficulties in directly computing the optimal value function through using arbitrary, for example, nonlinear or discontinuous, reward and transition functions. Different from that, the proposed approach makes the direct computation possible in practice, which opens a new line of work to solve stochastic motion planning problems.

Emerging as a popular sampling-based MPC approach, Model Predictive Path Integral (MPPI) control (Kappen et al., 2012; Theodorou et al., 2010; Theodorou et al. 2010, 2010; Williams et al. 2017, 2018) also exhibits relevance to our proposed method. This methodology has been developed specifically to address finite-horizon stochastic optimal control problems that conveniently allow arbitrary cost functions and nonlinear system dynamics. It shares a fundamental objective with our method, focusing on the resolution of the second-order Hamilton–Jacobi–Bellman (HJB) partial differential equation. Notably, MPPI correlates between noise and control costs, utilizing the Feynman–Kac formula to transform the HJB equation into a path integral formulation. This transformation enables MPPI to provide an MPC solution via a "forward" sampling strategy. Contrastingly, our method diverges in approach by leveraging a kernel representation of the value function. This representation forms the foundation for our methodology to directly compute solutions to the HJB equation via a policy iteration process. This differs our method from the MPPI framework and lays the groundwork for potential advancements and optimizations within the realm of stochastic optimal control problems.

3. Preliminary background

3.1. Markov decision processes

We formulate the robot decision-theoretic planning problem as an infinite-horizon discrete-time discounted MDP with continuous states and finite actions. It is defined by a 5-tuple

$\mathcal{M} \triangleq \langle \mathbb{S}, \mathbb{A}, T, R, \gamma \rangle$, where $\mathbb{S} = \{s\} \subseteq \mathbb{R}^d$ is the d-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 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, which specifies the one-step 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

$$v^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \middle| s_0 = s\right]. \tag{1}$$

The state at the next time step, s_{t+1}, draws from distribution $p(s_{t+1}|s_t, \pi(s_t))$. Let us use k to denote the computation epoch, then we can rewrite the above equation recursively as follows

$$v^{\pi}_{k+1}(s) = R(s, \pi(s)) + \gamma \mathbb{E}^{\pi}\left[v^{\pi}_k(s')\middle|s\right] \triangleq \mathcal{B}^{\pi} v^{\pi}_k(s), \tag{2}$$

where $\mathcal{B}^{\pi}$ is called the *Bellman operator*, and $\mathbb{E}^{\pi}[v^{\pi}_k(s')|s] = \int p(s'|s, \pi(s)) v^{\pi}_k(s') ds'$. The computation epoch k is omitted in the rest of the paper for notation simplicity. Equation (2) is called Bellman equation. The function $v^{\pi}(s)$ is usually called the *state value function* of the policy π. Solving an MDP amounts to finding the optimal policy π^* with the optimal value function which satisfies the *Bellman optimality equation*

$$v^{\pi^*}(s) = \max_{\pi}\{R(s, \pi(s)) + \gamma \mathbb{E}^{\pi}[v^{\pi}(s')|s]\}. \tag{3}$$

3.2. Approximate policy iteration via value function representation

Value iteration and policy iteration are the most prevalent approaches to solving an MDP. It has been shown that the value iteration and policy iteration can both converge to the optimal policy in MDPs with discrete states (Bertsekas 2012; Sutton and Barto 2018). Our work will be built upon policy iteration and here we provide a summary of the common value function approximation methods used in policy iteration when dealing with continuous states (Bertsekas 2011; Gordon 1999; Powell 2016).

Policy iteration requires an initialization of the policy (can be random). When the number of states in the MDP is finite, a system of finite number of linear equations can be established based on the initial policy, where each equation is exactly the value function (equation (2)). The solution to this linear system yields state-values for all states of the initial policy (Puterman 2014). 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 equation (1) for every state.

If, however, the states are continuous or the number of states is infinite, it is intractable to store and evaluate the value function at every state. One must resort to approximate solutions by finding an appropriate representation of the value function. 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, that is, equation (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 (Lagoudakis and Parr 2003). This obtained representation of the 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

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

where $\phi_i \in \Phi \triangleq \{\phi_1, \ldots, \phi_m\}$. The elements in the set Φ are called the *basis functions* in literature (Powell 2016), and these basis functions are usually parametric functions with a fixed form. A finite number of supporting states $\mathbf{s} = \{s^1, \ldots, s^N\}$, $N > m$ can be selected. The selection of supporting states $\mathbf{s}$ needs to take into account the characteristics of the underlying value functions. In the scenario of robot motion planning in complex terrains, it relates to the properties of the terrain, for example, geometry or texture. The solution to w^{π} can be calculated by minimizing the squared *Bellman error* over $\mathbf{s}$, defined by $\mathcal{L}(w^{\pi}) = \sum_{i=1}^{N}(v(s^i; w^{\pi}) - \mathcal{B}^{\pi} v(s^i; w^{\pi}))^2$. And w^{π} may have a closed-form solution in terms of the basis functions, transition probabilities, and rewards (Lagoudakis and Parr 2003). By policy iteration, the final solution for $v(s; w^{\pi})$ can be obtained. Note that $v(s)$ may also be approximated by any non-parametric nonlinear functions such as neural networks.

4. Methodology

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 (Boutilier et al., 1999; Puterman 2014), 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 4.1). 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 4.2). 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 4.3 and 4.4).

4.1. Taylored approximate Bellman equation

To design an efficient approach for solving continuous-state MDP problems, we essentially need to fulfill two requirements: *a suitable representation of the value function* and *efficient computation of the Bellman optimality equation*.

For the first requirement, we may directly apply the basis functions to approximate the value function and then solve the Bellman optimality equation. Yet this approach faces difficulties of explicitly specifying basis functions: if the set of basis functions is not rich enough, the approximation error can be large. A better approach may be a direct application of kernel methods to represent the value function (referred as the *direct kernel-based method*). We will discuss this method later. But this representation does not satisfy the second requirement, i.e., *efficient evaluation of the Bellman optimality equation*. This is because a fully specified transition probability function $p(\cdot|\cdot, \cdot)$ is required for computing the Bellman operator, but it is usually hard to specify in the real world. In addition, even this transition function can be obtained, the expectation $\mathbb{E}^\pi[v_k^\pi(s')] = \int p(s'|s,\pi(s))v_k^\pi(s')ds'$ generally does not have a closed-form solution, and computationally expensive numerical integration is typically entailed.

In contrast to directly solving equation (3) by value function approximation, we consider an approximation to the Bellman-type equation at first and then apply value function approximation. We approximate the Bellman 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 for most real-world applications. Additionally, we will see that evaluating the integration over the global state space is not needed. From this perspective, our approximation can be viewed as a local version of the original Bellman equation, i.e., it uses local gradient information to approximate the integral of the value function over possible next states.

Formally, we assume that the state space is of d-dimension and a state s may be expressed as $s = [s_1, s_2, \ldots, s_d]^T$. Suppose that the value function $v^\pi(s)$ for any given policy π has continuous first and second order derivatives (this can usually be satisfied with aforementioned value function designs). We subtract both hand-sides by $v^\pi(s)$ from equation (2) and then take Taylor expansions of value function around s up to second order (Braverman et al., 2020):

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

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

$$\left(\mu_s^\pi\right)_i = \int p(s'|s,\pi(s))\left(s_i' - s_i\right)ds', \qquad (6a)$$

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

for $i, j \in \{1, \ldots, d\}$; the operator $\nabla = [\partial/\partial s_1, \ldots, \partial/\partial s_d]^T$ and the notation $\cdot$ in the last equation indicate an inner product; the operator $\nabla\cdot\sigma_s^\pi\nabla$ is read as

$$\nabla\cdot\sigma_s^\pi\nabla = \sum_{i,j}\left(\sigma_s^\pi\right)_{i,j}\frac{\partial^2}{\partial s_i \partial s_j}. \qquad (7)$$

To be concrete, we take a surface-like terrain for example and use that surface as the decision-theoretic planning workspace, i.e., $s = [x, y]^T = [s_x, s_y]^T$. We have 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 equation (5) approximates calculation of equation (2) in the policy evaluation stage, the solution to equation (5) thus provides the value function approximation under current policy π. Equation (5) also implies that we only need to use the first $\left(\mu_s^\pi\right)_i$ and second $\left(\sigma_s^\pi\right)_{i,j}$ moments instead of computing the expectation of the value function using the original transition model $p(s'|s, \pi(s))$ to evaluate the Bellman operator equation (3) required in solving MDPs.

Note that our method can be naturally extended to include higher moments via higher-order Taylor expansions. We stop at the second moment because the first two moments are generally sufficient to describe the major characteristics of the transition function. Also, extending to higher moments will incur heavy computation which is a trade-off that we need to take into account.

4.2. Approximate Bellman optimality equation via diffusion-type PDE

Equation (5) is a diffusion-type partial differential equation (PDE). It is an approximation to the Bellman equation (8) through the second-order Taylor expansion. We will need to solve this PDE to obtain an approximation methodology to the Bellman optimality equation. Typically if solutions exit for a PDE, there could be infinite solutions to satisfy the PDE unless we impose proper boundary conditions (Evans 2010). Therefore, we need to analyze the necessary boundary conditions to equation (5).

In our problem settings, robots are not allowed to move out of free-space boundaries. We first observe that the value function should not have values outside of the feasible planning region, that is, the state space $\mathbb{S}$, and the value function should not increase towards the boundary of the region. Otherwise it will result in actions that guide the robot outside the free space. A practical boundary condition to impose can be that the directional derivative of the value function with respect to the outward unit normal at the boundary states is zero (see Figure 2 for an illustration). It is not the only condition that we can impose, but it is a relatively easier condition to obtain solutions with desired behaviors. Second, in order to ensure that the robot is able to reach the goal, we follow the conventional goal-oriented decision-theoretical planning setup and constrain the value function at the goal state to the maximum state-value in the state space $\mathbb{S}$. Consequently, these boundary conditions ensure that the policy does not control the robot outside of the feasible regions (safety) and also leads the robot to the goal area.

Formally, let us denote the boundary of entire continuous planning region by $\partial\mathbb{S}$ and the goal state by s_g. Suppose the value function at s_g is v_g. Section 4.1 implies that the Bellman optimality equation equation (3) can be approximated by the following second-order Hamilton–Jacobi–Bellman (HJB) PDE:

$$0 = \max_{\pi}\left\{\gamma\left(\left(\mu_s^\pi\right)^T\nabla v^\pi(s) + \frac{1}{2}\nabla\cdot\sigma_s^\pi\nabla v^\pi(s)\right) + R(s,\pi(s)) - (1-\gamma)v^\pi(s)\right\}, \tag{8}$$

with boundary conditions

$$\sigma_s^\pi\nabla v^\pi(s)\cdot\widehat{\boldsymbol{n}} \quad = 0, \quad \text{on}\,\partial\mathbb{S} \tag{9a}$$

$$v^\pi\left(s_g\right) \quad = v_g, \tag{9b}$$

where $\widehat{\boldsymbol{n}}$ denotes the unit vector normal to $\partial\mathbb{S}$ pointing outward, and condition (9a) constrains the directional derivative with respect to this normal vector $\widehat{\boldsymbol{n}}$ to be zero. The solution v^π of the above PDE can be interpreted as a diffusion process with μ_s^π and σ_s^π as drift and diffusion coefficients, respectively. We show a 2D example of this boundary condition in Figure 2. The normal vectors $\widehat{\boldsymbol{n}}$ at the boundary are perpendicular to the gradient of the value function $\nabla v^\pi(s)$, which constrains the value function from expanding outside the boundaries of the state space. In addition, the direction of each gradient vector points toward the goal, and this allows the policy to follow the value function gradient which guides the robot to move in the goal direction.

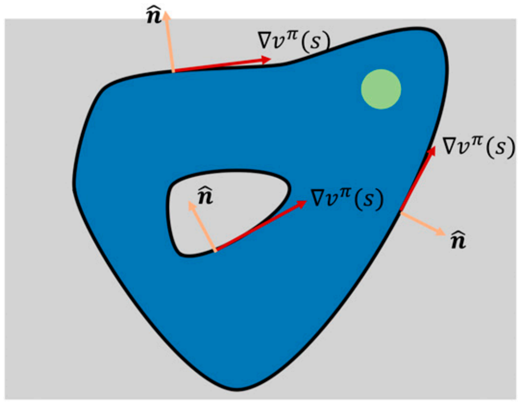
$\widehat{\boldsymbol{n}}$
$\nabla v^\pi(s)$
$\widehat{\boldsymbol{n}}$
$\nabla v^\pi(s)$
$\nabla v^\pi(s)$
$\widehat{\boldsymbol{n}}$

Figure 2. Illustration of the boundary condition in equation (9a) using a 2D example. The blue and green regions indicate the state space and the goal region, respectively. The gray areas represent the infeasible space, for example, obstacles, and the boundaries are shown as black curves. The normal vector $\widehat{\mathbf{n}}$ and the gradient of value function $\nabla v^\pi(s)$ at three arbitrary boundary points are indicated by yellow and red arrows, respectively.

The condition (9a) is a type of homogeneous Neumann condition, and condition (9b) can be thought of as a Dirichlet condition in literature (Evans 2010). This elegantly approximates the classic Bellman optimality equation by a convenient PDE representation. While equation (8) is a nonlinear PDE (due to the maximum operator over all the policies), our algorithm in the future section will allow to solve a linear PDE for a fixed policy. In the next section, we will leverage the kernelized representation of the value function to avoid the 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.

4.3. 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 equation (8) which allows us to extend with kernelized policy evaluation for Taylored value function approximation.

Specifically, let $k(\cdot,\ \cdot)$ be a generic kernel function† (Hofmann et al., 2008). 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

$$v^{\pi}(s') = \mathbf{k}(s', \mathbf{s})^T(\lambda\mathbf{I} + \mathbf{K})^{-1}V^{\pi}, \tag{10}$$

where $\lambda \geq 0$ is a *regularization factor*. When $\lambda = 0$, it links to the kernel ordinary least squares estimation of w^{π} in equation (4); when $\lambda > 0$, it refers to the ridge-type regularized kernel least squares estimation (Shawe-Taylor et al., 2004). Furthermore, equation (10) 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 equation (5) and boundary conditions equation (9a) and (9b).

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

$$\left(\mathbf{M}^{\pi}(\lambda\mathbf{I} + \mathbf{K})^{-1} - (1-\gamma)\mathbf{I}\right)V^{\pi} = \mathbf{R}^{\pi}, \tag{11}$$

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:

$$[\mathbf{M}^{\pi}]_{i,j} = \gamma\left(\left(\mu^{\pi}_{s^i}\right)^T\nabla_{s^i} + \frac{1}{2}\nabla_{s^i}\cdot\sigma^{\pi}_{s^i}\nabla_{s^i}\right)k(s^i, s^j). \tag{12}$$

Note that ∇_{s^i} indicates the derivatives with respect to s^i, that is, $\nabla_{s^i} = [\partial/\partial s^i_1, \ldots, \partial/\partial s^i_d]^T$. Here, we provide a concrete derivation of using the Gaussian kernel to our proposed kernel Taylor-Based approximate method as it is a commonly used kernel in practice and often used in the studies of kernel methods.

Gaussian kernel functions on states s' and s have the form $k(s', s) = c \times \exp((-1/2(s'-s)^T\Sigma_s^{-1}(s'-s))$, where c is a constant and Σ_s is a covariance matrix. Note that Σ_s is referred to as the lengthscale parameter in our work. The lengthscale governs the "smoothness" of the function, and a large lengthscale leads to a smooth function, whereas a small lengthscale causes a rugged function. 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 (e.g., equation (12)). In fact, we have

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

and

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

where $tr(\cdot)$ denotes the trace of the matrix. By plugging equations (13) and (14) into equation (12), we can obtain an expression for the elements in the matrix $\mathbf{M}^{\pi}$.

The solutions to the system equation (11) yield values of V^{π}. These values further allow us to obtain the value function (10) for any state under current policy π. This completes modeling our kernel Taylor-based approximate policy evaluation framework.

4.4. Kernel Taylor-based approximate policy iteration

With the above formulations, 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.

Algorithm 1 Kernel Taylor-Based Approximate Policy Iteration

```
Input: A set of supporting states $\mathbf{s} = \{\mathbf{s}^1, \ldots, \mathbf{s}^N\}$; 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. (10) 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. (11) in Section 4.1.
 6:     // Policy improvement step
 7:     for $i = 1, \ldots, N$ do
 8:         Update the action at the supporting state $s^i$ based
    on Eq. (15).
 9:     end for
10: until actions at the supporting states do not change.
```

Because our kernelized value function representation depends on the finite number of 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 improve the current policy at each support state

$$\begin{aligned}\pi_{k+1}(s) = \underset{a\in\mathbb{A}}{\operatorname{argmax}}\Bigg\{&R(s, a) +\\ &\gamma\left(\left(\mu^a_s\right)^T\nabla + \frac{1}{2}\nabla\cdot\sigma^a_s\nabla\right)v^{\pi_k}(s)\Bigg\},\end{aligned} \tag{15}$$

where $s \in \mathbf{s}$, π_k and π_{k+1} are the current policy and the updated policy, respectively. Note that μ^a_s and σ^a_s depend on a through the transition function $p(s'|s, a)$ in equation (6). Compared with the approximated Bellman optimality equation (equation (8)), equation (15) 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 $v^{\pi_{k+1}}(s) \geq v^{\pi_k}(s)$ (Bertsekas 2012). 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 equation (15).

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 (Si et al., 2004).

Intuitively, this proposed framework is efficient and powerful due to the following reasons: (1) by approximating the Bellman-type equation using the PDE, we eliminate the necessity in requiring a full transition function and the difficulty in computing the expectation over the next-state-values; (2) rather than tackling the difficulties in solving the PDE, 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 practical assumptions.

5. Algorithmic performance evaluation

Before testing the applicability of our proposed method in real-world environments, we conducted algorithmic evaluations in two sets of simulations in order to validate the efficiency of the proposed framework.

To do so, we eliminate the real-world complexities that are not the focus of the main method (e.g., perception, partial observability, and online re-planning). The first evaluation (Section 5.1) is a goal-oriented planning problem in a simple environment with obstacle-occupied and obstacle-free spaces. In the second evaluation (Section 5.2), we demonstrate that our method can be applied to a more challenging navigation scenario on Mars surface (Maurette 2003), where the robot needs to take into account the elevation of the terrain surface (i.e., "obstacles" become continuous and are implicit).

5.1. Plane navigation

5.1.1. Task setup. Our first test is a 2D plane navigation problem, where the obstacles and the goal area are represented in a $10m \times 10m$ environment, as shown in Figure 3. The state space for this task is a 2-dimensional Euclidean space, that is, $s = [s_x, s_y]^T$ and $s \in \mathbb{S} \subseteq \mathbb{R}^2$. The action space is a finite set with a number of Q points $\mathbb{A}(s) = \{a_i(s) | i \in \{1, \ldots, Q\}\}$. Each point $a_i(s) = [s_x + r\cos(2\pi i/Q), s_y + r\sin(2\pi i/Q)]^T$ is an action generated on a circle centered at the current state with a radius r. In this evaluation, we set the number of actions $Q = 12$ and the action radius as $r = 0.5\ m$. An action point can be viewed as the "carrot-dangling" waypoint for the robot to follow, which is the input to the low-level motion controller.

We use a Gaussian distribution as the transition function. The mean of Gaussian represents the selected (intended) next waypoint, while the variance is set to $0.1\ m$ in both x and y axes, accounting for the error in the low-level controller when executing the waypoint command. We use a sparse-reward setting, where when the agent arrives at the goal and obstacle, it receives $+1$ and -1, respectively. This setting is known to be challenging for most approximate MDP methods like deep RL to solve (Nair et al., 2018a).

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, that is, the robot cannot transit to any other states if they are in these states. To satisfy the boundary condition mentioned in Section 4.2, 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.

5.1.2. 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 an infinite number of states, we numerically evaluate the quality of a policy using the average return criterion (Islam et al., 2017). In detail, we first uniformly sample 10^4 states to ensure a thorough performance evaluation. Then, for each sampled state, we execute the policy for multiple trials, that is, generating multiple trajectories, where each 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.

5.1.3. Baseline setup for comparison. The performance of the proposed method is compared against four baseline approaches. The first baseline is *direct kernel-based policy iteration* (Kuss and Rasmussen 2004). It approximates the value function using the kernel method with the traditional Bellman update (equation (2)) that requires the full transition probability. For a fair comparison, we set the kernel lengthscale as 1, which is the same as our method.

We also compare with the *grid-based policy iteration* which is our second baseline, where the continuous state space is discretized into grids, and the vanilla policy iteration is used to solve the discretized MDP by iterating over all the grids and actions.

The third technique, *neural-fitted value function for policy iteration (N-FVPI)*, represents a class of value-based RL methods, where a neural network is used to represent the value function $v^{\pi}(s)$ to handle the continuous state space (Heess et al., 2015). During the policy evaluation step, the value function's parameters are optimized to reduce the one-step squared Bellman residual via gradient descent (Lutter et al., 2021). Like the previous approaches, the policy is implicitly derived by selecting an action that maximizes the

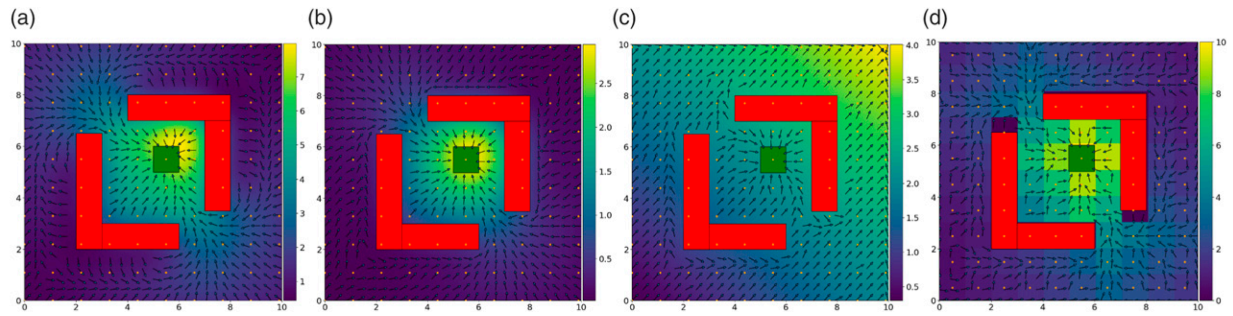

Figure 3. Evaluation with a traditional simplified scenario where obstacles and goal are depicted as red and green blocks, respectively. We compare the final value function and the final policy obtained from (a) kernel Taylor-based PI, (b) direct kernel-based PI, (c) N-FVPI, and (d) grid-based PI. A 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 the states, or the grid centers (in the case of grid-based PI), which are used to update the value functions.

Bellman equation in equation (3). In the experiment, we use a shallow two-layer network with 100 hidden units in each layer and a SiLU activation function (Elfwing et al., 2018).

To ensure a comprehensive comparison, besides the above-mentioned value-based methods, where the policy is implicitly derived from the value function, we also evaluate our method against the state-of-the-art actor-critic method, that is, *proximal policy gradient* (PPO) (Schulman et al., 2017). It was selected as a strong baseline from the actor-critic family due to its proven track record in training complex control policies for various challenging robotics platforms, including legged locomotion (Miki et al., 2022) and high-speed drone racing (Kaufmann et al., 2023). In the experiment, the value function and the policy are approximated using neural networks with the same configuration as N-FVPI. Additionally, PPO generally uses zeroth-order gradients, that is, REINFORCE-type policy gradient (Sutton and Barto 2018), to optimize the policy parameters. In contrast, the value-based methods, that is, grid-based, direct kernel-based, and N-FVPI, only compute the value function, and the policy is derived implicitly via the Bellman optimality equation.

5.1.4. Results. The comparison of the above methods aims at investigating three important questions:

1. In this sparse-reward navigation problem, how does the kernelized value function representation compare to alternative representations such as neural networks or grid-based methods?
2. In contrast to our approach, 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 perform similarly to the direct kernel-based method?
3. How does the performance of our method compare to the state-of-the-art actor-critic (reinforcement learning) method, that is, PPO?

The results for all five methods are shown in Figure 4 on the average return with respect to the number of states used to update the value and policy functions. The first question is answered because, among the value-based methods, the kernel method (our kernel Taylor-based and direct kernel-based PI) consistently outperforms the other two approximators (neural network and grid based). Moreover, the second question can be answered by observing that our method has a 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 complete knowledge of transition functions. In contrast to grid-based PI, kernel-based algorithms and N-FVPI can achieve moderate performance even with a small number of supporting states. It implies 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 N-FVPI. Lastly, the actor-critic method, PPO, requires more samples to achieve performance on par with the value-based methods, but it exceeds the performance of the N-FVPI after using 100 samples to update the value and policy networks. However, due to the complex approximators used (neural networks), PPO requires more samples to achieve performance on par with the kernelized value function representation.

In Figure 5, we compare the computational time and the number of iterations to convergence for all the value-based methods. The computational time of our method is less than that of the grid-based method, as revealed in Figure 5(a). We notice a negligible computational time difference between our and direct methods. As a parametric method, N-FVPI has the least computational time and increases only linearly, but it does not converge as indicated by Figure 5(b).

The function values and the final policies are visualized in Figure 3. All methods except for the N-FVPI obtain reasonable approximations of 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 not smooth.

5.1.5. Impact of hyperparameters. We also examine the impact the hyperparameters, for example, number of supporting states and lengthscales, on the method's performance.

To achieve this goal, we place evenly-spaced supporting states (in a lattice pattern) with different spacing resolution. Besides the number of states, the kernel lengthscale 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 Figure 6. The lengthscale and regularization parameters are searched over a set of values whose range is pre-estimated by the workspace dimensions and obstacle configurations, $\{0.5, 1, 1.5, 2, 2.5, 3\}$. By entry-wise comparison among the four matrices in Figure 6, 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 (Figure 6(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 into selecting 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 4.3). On the other hand, high-performing policies (indicated by yellow) appear more at the bottom of the performance matrix when a greater number of supporting states are present, which means that a smaller length scale is generally required given a larger quantity of supporting states.

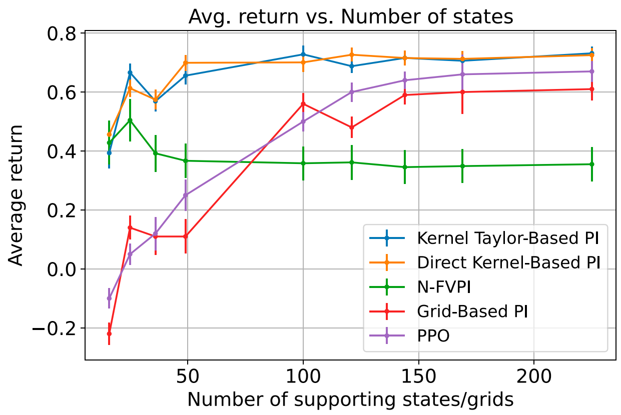

Figure 4. The comparison of the average return of the policies computed by our method (kernel Taylor-based PI) and four other baselines. The *x*-axis is the number of supporting states/grids used in computing the policy. In the case of PPO, it represents the number of samples used for value and policy updates. The *y*-axis shows the average return. The error bars represent the standard deviations.

5.2. Martian terrain navigation

In this evaluation, 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) (McEwen et al., 2007). 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 a probability proportional to the slope angle. Otherwise, its next state is distributed around the desired waypoint followed by the current action. This indicates that the underlying transition function should be the mixture of these two factors, and it is reasonable to assume that the means of the two cases are given by the current state and the next waypoint, respectively. We can similarly have an estimate of the variances. The transition function's mean and variance 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

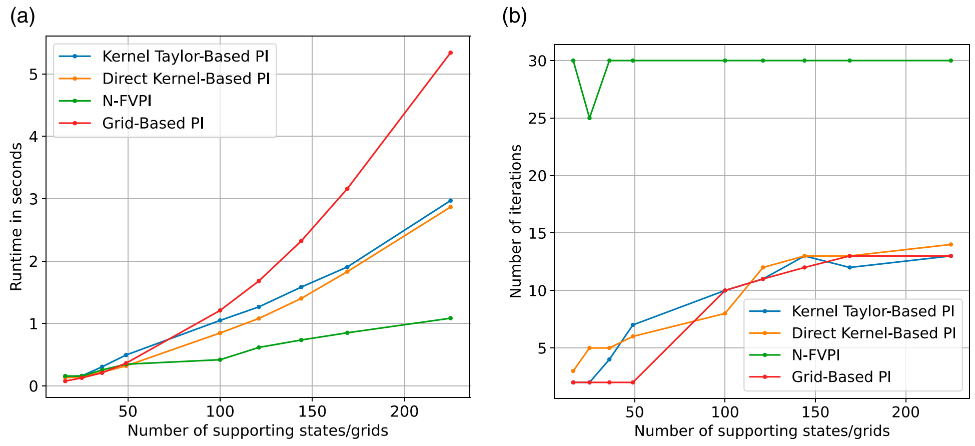

Figure 5. Computational time comparisons of the four value-based algorithms with changing number of states. (a) The computational time per iteration and (b) number of iterations to convergence.

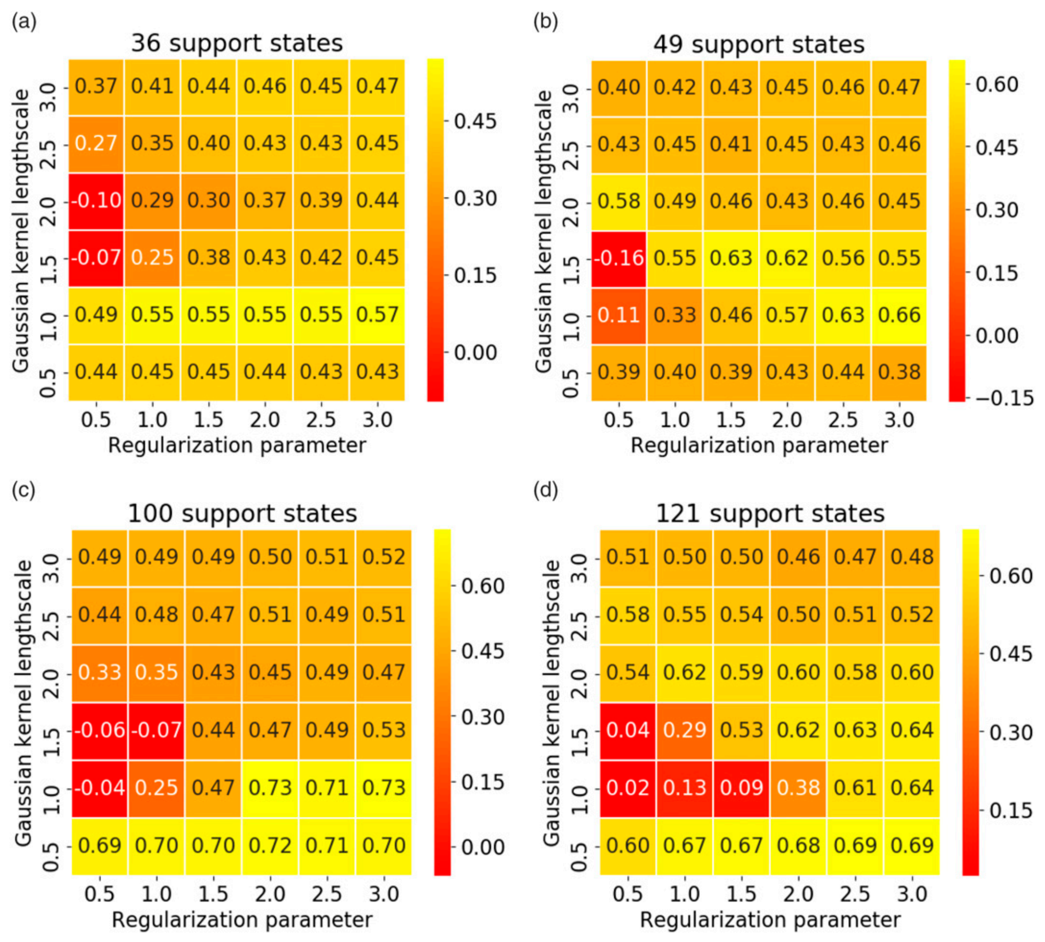

Figure 6. The performance matrix obtained by the hyperparameter search using (a) 6 × 6, (b) 7 × 7, (c) 10 × 10, and (d) 11 × 11 evenly-spaced supporting states. Rows and columns represent different Gaussian kernel lengthscale and regularization parameters, respectively. The numbers in the color map represent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

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.

Figure 7(a) and (c) compare the two methods of differing 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 which the rover needs to plan paths to reach 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 Figure 7(d) attempt to approach the goal (red star) with minimum distances, and at the same time, avoid dangerous terrains by choosing more leveled/even surfaces. In contrast, the trajectories obtained using the evenly-spacing states in Figure 7(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 Figure 8(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 Figure 8(a) where the background color map denotes the elevation of terrain.

6. Autonomy system design

Previous algorithmic evaluations assume that a well-defined MDP can be abstracted and constructed from the real-world problem and focus on algorithmic property/performance assessments. However, in reality, specifying an MDP for stochastic motion planning problems requires a known task region for constructing the state space as well as detailed information on the environment for transition and reward functions. These requirements are generally very difficult to satisfy if we deploy the robots into the real world, especially in the off-road environment where the task region is hard to define and no prior map is provided. In such cases, the robot needs to use its onboard sensors to acquire information and make decisions online. Specifically, it needs to process the observations, construct a new MDP within the observed area, compute the corresponding policy, and execute it. This process should repeat until the task is completed. In this paper, we only focus on generating a local policy within the observable area. Global planning in any unknown environment might require information-driven (active) sensing to achieve a trade-off between the exploration of unknown space and the exploitation of observed space, which, however, is not the focus of this work.

Furthermore, we confine the MDP construction process to only relying on the environments' geometric information and develop the navigation system based on it. This capability is vital if only perception depth information is available (e.g., with point cloud from any ranging sensors) through which we can leverage mostly the reconstructed geometry. As illustrated in Figure 1, the uneven terrains pose many challenges (e.g., traversals of hills, ridges, valleys, slopes, etc.) as one of the primary features affecting the robot's motion efficiency is geometry.

In this section, we present our autonomy system that extracts the MDP elements (supporting states, transition function, and reward function) from depth sensor data and connects the perception to the action loop. Figure 9 provides an overview of the system.

6.1. Perception

The perception module is responsible for processing sensor observation and providing information for planning. We use LiDAR to acquire geometric information about the environment. Based on the point cloud, it generates the robot state (pose) and a map of the environment in the field of view. The pose information can be estimated by any existing LiDAR localization methods (e.g., Bresson et al. (2017)). For the map representation, we utilize the cost map to provide occupancy information for identifying obstacle-free regions for navigation. However, there exist scenarios in

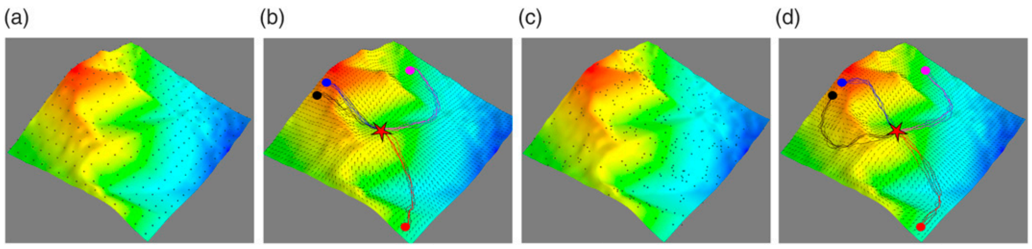

Figure 7. Supporting state distributions and the policies for evenly-spaced selection and importance sampling-based selection. The 3D surface shows the Mars digital terrain model obtained from HiRISE. Supporting states and policies are shown in black dots and vector fields. The colored lines represent sampled trajectories, which initiate from four different starting positions indicated by the circles with the same colors. (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.

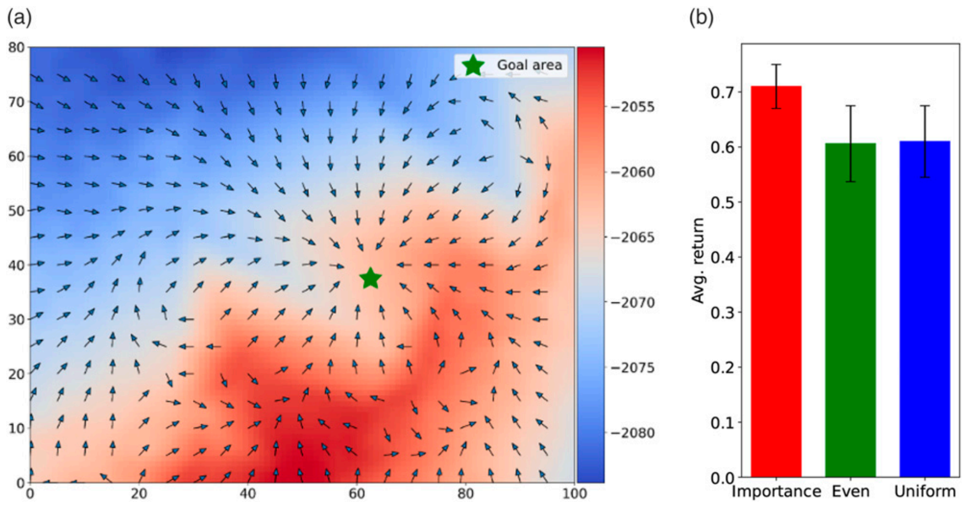

Figure 8. (a) The top-down view of the Mars terrain surface as well as the policy generated by our method with the importance sampling selection. The color map indicates the height (in meters) of the terrain. (b) The comparison of average returns among three supporting state selection methods using the 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.

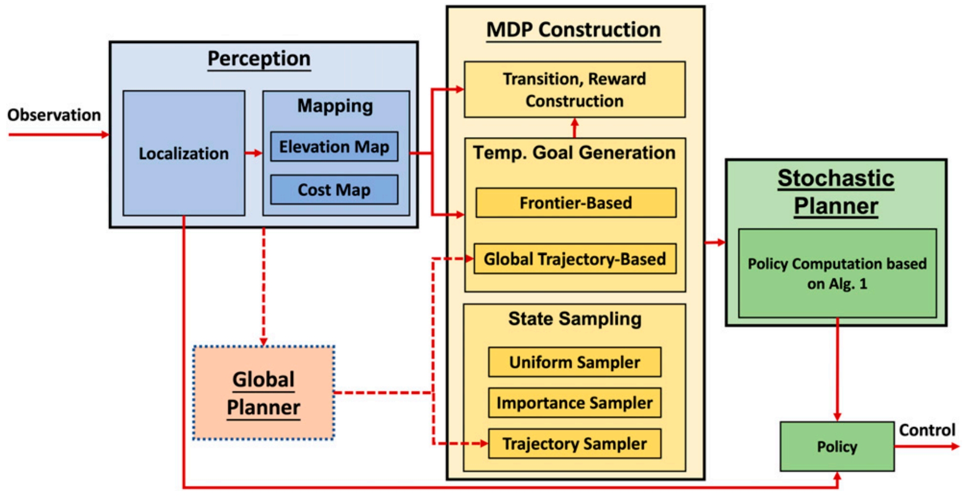

Figure 9. An overview of the system for navigation in prior-unknown environments. The red arrows represent the direction of the information flow. The global planner module in the dashed block is optional.

uneven terrains where it might inaccurately model the *navigability* of certain regions. For instance, areas with considerable elevation might be misrepresented as occupied, despite the possible traversability due to a non-steep gradient of the surface. Thus, we use a $2.5D$ elevation map (Fankhauser et al., 2018) to provide a finer depiction of the terrain shape. Combined with the variance design based on the slope information described in the previous section, the robot can reason about navigability beyond mere occupancy.

6.2. MDP construction

The second part of the system builds an MDP using the mapping information. In the following, we explain this construction process for each MDP element.

6.2.1. State and action spaces. The state and action spaces are pre-defined with the given vehicle type (ground). Specifically, the state is represented as the robot pose with position and orientation information (x, y, θ), and the action consists of linear and angular speeds (v, ω). Since our method deals with the discrete action space, we can tessellate the continuous action into evenly-spaced intervals (i.e., equi-distanced linear/angular speed levels). The discretization resolution, as well as the minimum and maximum values for the speeds, depend on the robot's capability and the task's complexity.

6.2.2. Reward function. When using the cost map, the reward function consists of two parts $R(s) = (1 - \mathbb{I}_{s_g}(s))c(s) + r_g\mathbb{I}_{s_g}(s)$. The first part $c(s)$ is the obstacle penalty provided by the cost map, $\mathbb{I}_{s_g}(s)$ is an indicator function for checking whether the state belongs to the goal region s_g, and r_g is the goal reward. If only the elevation map is used, the obstacle penalty is not included, and the robot only receives a positive reward if it arrives at the goal. Since we focus on planning within the observable space only, we need to generate a dummy goal region s_g in the space of the field of view. In other words, it needs to set a temporary goal if the final goal region is outside of the robot's current observed area. We use two heuristic approaches to generate this temporary goal. The first one is inspired by the frontier-based concept which generates a temporary goal chosen at the boundary of the current observable map and with the least distance towards the final goal (Burgard et al., 2005; Yamauchi 1997). The second method utilizes the path generated by a global planner if available (usually just for high level guidance). The temporary goal can be determined by "cropping" a path segment from the global path within the field of view only, where the path segment starts from the current robot pose and extends to a pre-defined length along the cropped path, and the final pose on the extracted path segment serves as the temporary goal. Note that such a global planner is not mandatory for our method as it can be replaced with any frontier-based goal selection method.

6.2.3. First two moments of the transition function. We assume that the robot motion is generated based on $s' = f(s, a, h) + \epsilon$, where $f(s, a, h)$ is a deterministic function representing the discrete-time motion model related to vehicle dynamics, for example, a differential drive model; $h(s)$ is the slope angle at state s derived from the elevation map; and ϵ is a noise term independent of the state and action. If the elevation map is used, the motion model considers the effect of the elevation on the robot's motion. Otherwise, we treat $h(s) = 0$ (flat surface) for all states in the current state space. To construct $f(s, a, h)$, we modify the Dubin's car model to take into consideration of the terrain slope, $x' = (\pi/2 - h(s))(x + v\Delta t \cos\theta) + h(s)x$, $y' = (\pi/2 - h(s))(y + v\Delta t \sin\theta) + h(s)y$, and $\theta' = \theta + \omega\Delta t$. Here, $h(s) \in [0, \pi/2]$ is the slope angle at state s and Δt is the time discretization resolution. Intuitively, this model penalizes the distance traveled on the surface with a larger slope angle. Based on this formulation and equation (6), the first and second moments can be computed as $\mu(s, a, h) = \Delta_a s + \mathbb{E}[\epsilon]$ and $\sigma(s, a, h) = \mathbb{V}[\epsilon] + \mu(s, a, h)\mu(s, a, h)^T$, where $\Delta_a s = f(s, a, h) - s$ is the state shift after applying action a. In this work, we choose the mean $\mathbb{E}[\epsilon] = 0$ and variance proportional to the slope $\mathbb{V}[\epsilon] = kh(s)$, where $k \geq 0$ is a constant that modulates the impact of the slope on the model's uncertainty. Specifically, a larger value of k indicates increased uncertainty in the robot's motion on elevated terrains. This variance formulation allows the planner to be more cautious about navigating on high-slope terrain surfaces, enabling the robot to avoid these hazardous areas. The advantage of our method is obvious from the above modeling perspective. Since we only need to model the mean and variance of the noise ϵ, it is not necessary to acquire the exact probability distribution of the noise.

6.2.4. Supporting states. The last part of the construction is state sampling, which is responsible for distributing the supporting states within the state space. The positions and the number of supporting states are critical as they determine the accuracy of the value function and also computational time. More supporting states generally provide a better estimation of the optimal value function but require more time to compute as shown in Section 5.1. We consider uniform sampling, importance sampling, and trajectory sampling strategies in this work. All these sampling methods require a region to distribute the supporting states. The first two methods introduced in Section 5.1 and Section 5.2 need prior knowledge of the environment and specification of a sampling region, for example, a rectangular workspace. In greater detail, the uniform sampling tessellates a pre-defined region into equal-sized cells and uses the cell vertices as the supporting states. The importance sampler first uniformly samples a large number of states and then selects them based on some weighting criteria, for example, the slope of the terrain at a given state point. The trajectory sampler utilizes the global path generated by a global planner as a heuristic to define the planning and sampling region. It extracts a path segment on the global path starting from the current robot pose and distributes the state samples around this path segment, whose length can also be determined by the maximum linear speed of the robot. It distributes the state samples around the same path segment extracted from the global trajectory-based goal generation method. This method is especially useful when fast online computation is required since it does not need to search the entire planning region.

6.3. Policy computation and execution

With all the MDP elements ready, we can use Alg. 1 to compute a policy for the constructed MDP. Then, this policy is used for generating control actions $a = (v, \omega) = \pi(s)$. It is necessary to discuss two motion planning and control strategies of this system based on the global planner's ability to plan an executable path on uneven terrains.

6.3.1. Periodic re-planning. Conventional commonly-used global path planners (e.g., sampling-based or graph-based methods) generally do not consider the tracking ability of the lower-level planner and controller. As a result, when the low-level controller cannot execute the planned path accurately, the robot may deviate and fail to complete the task. This issue is particularly prevalent when the robot navigates on highly unstructured uneven terrains, where behaviors such as slippage or other unpredictable motion outcomes can occur frequently. In contrast to the traditional path planning framework that calculates a path while disregarding the inherent uncertainty when executing the global path, the proposed approach computes a feedback policy across a specified region by considering the possible errors that the low-level controller can easily make due to possibly fast-varying terrain elevations. In essence, our proposed method synthesizes the entire process of planning and control under uncertainty within a single framework.

We use the frontier-based exploration method (Yamauchi 1997) to generate temporary goal points. It is worth mentioning that when the planning region is large, a high computational load can naturally occur, resulting in a pausing behavior when the robot needs to replan. This can be mitigated and tuned by reducing the cropped planning region. Once the policy is computed within the selected region, it can query the action anywhere in the defined state space in real-time, as action computation only requires iterating over a finite number of actions. Also, re-planning is typically invoked periodically, initiated either when the robot reaches a predetermined interim goal or when new map information necessitates modifying the current policy, such as when the initially planned policy leads to a collision. Upon either event, the robot ceases its current policy and recalculates a new one incorporating this updated information. While this approach might cause delays due to re-planning over a large region, it facilitates effective robotic operation in situations where the path determined by the conventional global planner is difficult for the low-level controller to track.

6.3.2. Real-time planning with a global trajectory. If the global planner can provide a reasonable trajectory, which incurs only moderate tracking error, this information can reduce the search space by using the trajectory sampling strategy to obtain states around the globally planned path. The region encompassed by these states may be conceptually viewed as a tube within which the robot must remain, similar to the funnels constructed in Majumdar and Tedrake (2017). Focusing the policy search around the global path allows MDP construction and policy iteration to be performed in real-time. Subsequently, this affords the implementation of the Model Predictive Control (MPC) paradigm for real-time execution. After each policy iteration computation, the robot executes a singular action from the feedback policy and initiates a re-planning process.

7. Experiments using a realistic physics simulator

7.1. Experimental setup

To accomplish the autonomous navigation system described above, we first integrate the perception module and test the system's performance. The robot is no longer provided with a prior map of the environment, and it can observe the environment and obtain state information from its onboard depth sensor.

Our test scenario is designed to assess the navigation efficiency of the robot within a high-fidelity unity simulation environment, where the robot is required to navigate without a prior map. The simulator simulates a ClearPath Warthog differential drive ground vehicle with a 64-beam LiDAR. Its task is to navigate to a set of pre-defined waypoints sequentially and return to the initial position. The snapshot of the simulator environment and a bird-eye map view can be observed in Figure 10. This simulation environment encapsulates a range of challenging characteristics typically present in real-world off-road settings. Among these challenges are the uneven and textured ground terrain, clusters of densely packed obstacles, and narrow alleyways, which all pose substantial complexities for effective navigation. In addition, our mission definitions require the robot to plan and navigate areas without clearly distinguishable dirt-roads. These aspects of the environment necessitate a planning and motion control method that is precise enough to navigate constricted passages and robust enough to accommodate disturbances arising from uneven surfaces.

We set the robot's minimum and maximum values for linear and angular speeds as $v_{min} = -1\ m/s$, $v_{max} = 1\ m/s$, $\omega_{min} = -1.5 rad/s$ (turning right), and $\omega_{max} = 1.5 rad/s$ (turning left). In this experiment, we test the real-time local planning capability of the system, where an anytime version of A* (ATA*) (Likhachev et al., 2005) is employed as the global planner to provide the desired reference trajectory, and we sample support states around it, as described in Section 6.2.4.

We compare our method with the nonlinear model predictive control (NMPC) technique for path optimization. NMPC is a deterministic local trajectory optimization approach (see Allgöwer and Zheng (2012) for an overview), and it has a track record of documented successes in path planning over real-world challenging terrains (Gregory et al., 2016; Howard and Kelly 2007). The NMPC is still benchmarking for highly agile trajectories in cluttered and challenging environments (e.g., drone flight at speeds up to 20 m/s (Sun et al., 2022)). The traditional three-layer planning and control system often adopts this pipeline. We implemented the NMPC which produces smooth trajectories that respect the robot's dynamics by optimizing a cost function penalizing the path deviation from a desired global trajectory using algorithms from the NLOPT library (Johnson).

The two approaches were tested under three settings: 10, 5, and 4 waypoints, respectively. Figure 11 shows the waypoint positions, the constructed occupancy map, and two sampled trajectories for each setting using our method. The action space for our method is discretized evenly into 8 × 8 intervals which are dense enough to approximate continuous actions. For both methods, we choose $6m$ as the length of trajectory sampling, meaning that the robot plans $2s$ in the future if it uses the maximum speed 3 m/s.

7.2. Metrics

Both methods are evaluated with the following four metrics:

- *Tracking Error*: This metric measures the averaged distance between the global path and the executed path in the (x, y) axes. A smaller tracking error indicates the method can better execute the plan with a smaller deviation from the global path and is preferred.
- *Average Jerk (Avg. Jerk)*: This metric indicates the robot's average sum of acceleration changes along (x, y) directions across a mission. Generally, a smaller value reflects a smoother motion execution of the mission and is preferred.
- *Completion Time*: The completion time is the total traversal time to arrive at all the designated waypoints. It is computed over the successful runs only.

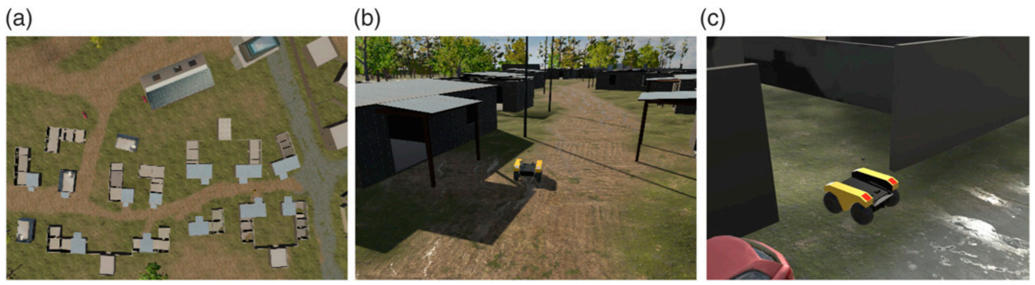

Figure 10. The unity simulation environment. (a) The top-down view of the environment, (b) a close view of navigable space in the simulator, and (c) an example of the narrow alleyway in the environment.

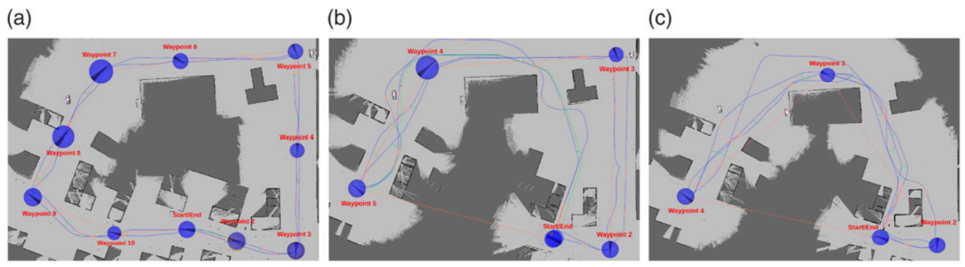

Figure 11. The mission maps and sampled robot trajectories for (a) ten, (b) five, and (c) four waypoint settings. The blue circles are the waypoints. For each setting, the robot starts from the initial position, navigates sequentially to the next points, and returns to the initial position. The blue paths represent the trajectories in two runs using our method. The straight red lines between waypoints show the mission's connection (i.e., order of waypoints). The other colored short path segments are related to robot's simultaneous localization and mapping (SLAM). In particular, the short red line segments scattered along the traveled path correspond to the pose graph factors after the successful execution of iterative closest point (ICP) registrations (Grisetti et al., 2010). Instances of successful loop closures are demonstrated by green lines linking two navigated paths. Lastly, the extended light green lines in (a) (b) and (c) represent the global plan.

- *Success Rate*: The success rate indicates the number of times the robot arrives at all the waypoints out of 10 runs without collision or getting stuck.

Compared to the averaged cumulative rewards that we used for evaluating the basic algorithmic performance (Section 5.1.2), these metrics are more relevant to the robot navigation task. We record these metrics only during the execution of the optimized trajectory (or policy), which excludes the global planner's influence on the evaluation.

7.3. Result

The result is shown in Table 1. In general, our method is more efficient and can complete the task using a shorter traversal time than NMPC. Additionally, our method completes all 10 runs for the three waypoint settings, while NMPC achieves 100% success rate only in the 10 waypoint setting (i.e., with sufficient waypoint guidance). Since the NMPC method attempts to follow the global path exactly, the trajectory optimized by NMPC sometimes oscillates around and overshoots the global path. This oscillation causes the vehicle to reduce its speed due to frequent turning. In contrast, since our method considers the uncertainty (via the second moment) of the robot's motion, it can reason about a region around the global path. As a result, as long as the robot remains within the area supported by the sampled states, it does not show oscillation behavior. The second row (average jerk) of the table can reflect this comparison, where in the 5 and 4 waypoint scenarios, our method produces a smaller jerk, resulting in smoother trajectory execution, while NMPC needs to stop abruptly when it overshoots far away from the global path. We can also observe that the global path overshooting problem of NPMC leads to larger tracking errors in the last two waypoint settings.

It is also worth pointing out that the robot uses a shorter time to complete the task in the 4 waypoint mission than in the 10 waypoint. This can be explained by observing the trajectories the robot generated in the two missions in Figure 11. Although the 10 waypoint mission provides denser waypoints, the robot is restricted to following these pre-defined points, which are not necessarily the shortest route. In contrast, the 4 waypoint mission has less restriction on which trajectory the robot should follow. As a result, the robot chooses a shortcut to navigate from Waypoint 2 to Waypoint 3 as shown in Figure 11(c).

8. Real-world demonstrations in indoor cluttered and outdoor unstructured environments

To demonstrate the applicability of our system in the real world, we conducted extensive real-world trials in different scenarios. The goal is to validate whether the proposed method can be used to generate effective policies that navigate a ground vehicle in complex and unstructured environments. The videos of the robot behaviors are demonstrated in Extensions 1 and 2.

8.1. Hardware setup

We implement our system on two ground vehicles, ClearPath Jackal and Husky. Since Jackal is small in dimension and has a small wheel traction, it is only used for indoor experiment. It is equipped with a 16-beam Velodyne LiDAR for localization and mapping, a quad-core 2.7 GHz CPU and 16 GB RAM for the onboard computation. Husky is used for navigating in outdoor environments because it is larger in size and provides more traction when moving on rough terrains. Husky is equipped with a 64-beam Ouster LiDAR for more detailed terrain mapping and a Lord Microstrain 3DM IMU for more accurate localization outdoor. It has an eight-core 2.7 GHz Intel I7 CPU and 64 GB RAM for the onboard computation.

Table 1. Comparison of diffusion MDP local planner (our method) and NMPC in different scenarios. Best-performing metrics are highlighted in bold. The statistics are averaged over 10 runs.

	10 waypoints	5 waypoints	4 waypoints
Diffusion MDP			
Tracking error (m)	0.43 ± 0.09	**0.56 ± 0.14**	**0.51 ± 0.11**
Avg. jerk (m/s^3)	6.73 ± 0.86	**6.35 ± 0.62**	**6.57 ± 0.34**
Completion time (s)	**157.2 ± 1.3**	**198.6 ± 11.2**	**177.8 ± 9.3**
Success rate	**10/10**	**10/10**	**10/10**
NMPC			
Tracking error (m)	**0.41 ± 0.14**	0.63 ± 0.11	0.69 ± 0.16
Avg. jerk (m/s^3)	**5.9 ± 0.56**	6.77 ± 0.41	7.14 ± 0.39
Completion time (s)	161.5 ± 4.7	204.6 ± 4.1	203.3 ± 3.5
Success rate	**10/10**	8/10	4/10

8.2. Indoor cluttered environment

The setup is an indoor 5 m^2 environment with random boxes as obstacles, and the task is to navigate the ClearPath Jackal to a goal location and return to the start position. The first aim is to validate the basic obstacle-avoidance behavior in a prior-unknown environment. The second aim is demonstrating that our system can achieve efficient navigation behavior without a global planner. We leverage the elevation representation for the MDP construction. Traditionally, the obstacle penalty in the reward (or cost) function is used as an indirect method for expressing the potential consequences of collisions. In this experiment, we also demonstrate that we can achieve the same obstacle-avoidance behavior without the obstacle penalty, using only the two statistical moments of the robot's motion based on the elevation map (see Section 6.2). This prepares us for navigation on uneven terrains demonstrated in the next section, where obstacle boundaries may be unidentifiable and the traditional use of the cost map may be inadequate.

Since the planning region is pre-specified (note that it does not mean the environment map needs to be provided a-prior), we evenly distribute 6 supporting states in each of the x, y dimensions, so that the distance between two neighboring states is $1m$ in each dimension. However, because it is difficult to predict what heading angles the robot may take during the task execution, we need supporting states to cover one full rotation of the heading angle, that is, $[-\pi, \pi)$. Specifically, we place 10 equidistant states in the θ dimension so that the distance between two neighboring states along the θ dimension is $\pi/5$. The total number of states is 360.

The range of linear and angular speeds are set to $v \in [-0.5\ m/s,\ 1.5\ m/s]$ and $\omega \in [-1.5 rad/s,\ 1.5 rad/s]$, respectively. The environment, the robot's motion sequence, and detailed planning information are shown in Figure 12. The initial position of the robot is placed in front of the obstacles, and the goal is between the two obstacles and is indicated by the green square on the second row of Figure 12. Initially, due to the occlusion by randomly placed boxes, the robot can only build a partial elevation map using the observed points. Then, it uses this initial map to calculate the value function within the state space. The value function is shown as the upper $2.5D$ surface in the second row. To aid the visualization, we only draw the value function over the position variables at the current robot's angle, and the height of the surface represents the state value. We can observe that the value function correctly characterizes the environment's geometry and the task. Specifically, the goal region has the highest state value. Since the policy always selects actions that maximize the state value, executing the policy from this value function will allow the robot to converge at the goal point. Additionally, on the value function surface, the narrow ridge between the two obstacles reflects the navigable space correctly. As a result, the policy derived from the value function can navigate the passage safely, as shown in the first row of Figure 12. As the robot executes its planned policy, it receives more sensing points and gradually builds a more detailed elevation map. This progressive refinement of the elevation map can be seen from the second and fourth rows. After the robot arrives at the goal, it re-plans and returns to the initial position, which is shown in the third row. As the robot's forward maximum speed ($v_{max} = 1\ m/s$) is larger than its backward maximum speed ($v_{min} = -0.5\ m/s$), it strategically reverses and aligns its forward axis with the initial position to ensure leveraging the maximum linear speed (the behavior can be observed in the supplementary videos). This process shows that the computed value function allows the robot to reason about its under-actuated dynamics through the two moments. This also reveals that our method exhibits a *speed-adaptive* behavior for the most efficient motion.

8.3. Outdoor unstructured environment

We further test our system in two outdoor environments shown in Figures 13 and 14. These environments contain unstructured terrains and irregular obstacles that introduce not only challenges in generating the obstacle-avoidance behavior but also difficulties in maintaining the robot's stable motions while traversing uneven/rough surfaces.

8.3.1. Navigation on uneven terrains. Figure 13 illustrates the robot's performance of navigating outdoor

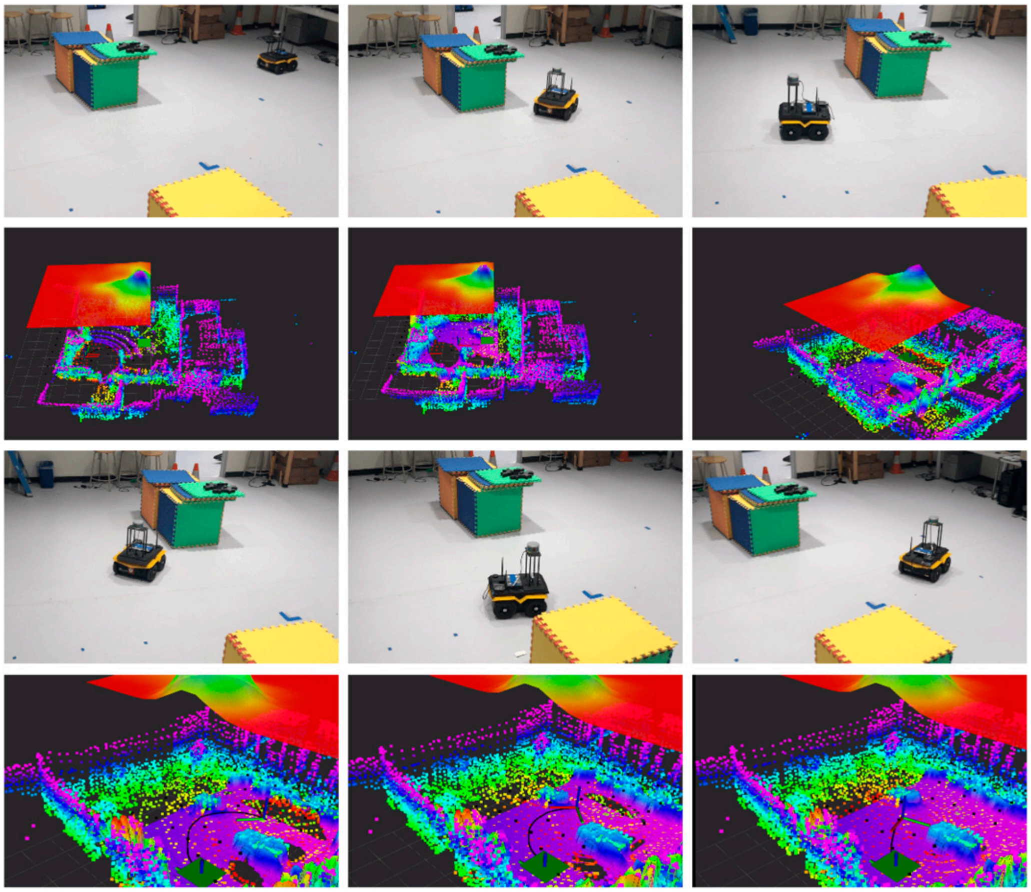

Figure 12. The indoor environment where our method is tested for validating its basic obstacle-avoidance and goal arrival behaviors. The first and third rows show the vehicles' real-world behaviors at different timesteps, and the second and fourth rows depict the corresponding visualization results. The task for the robot is to navigate through the gap between boxes (approximately 1.5 m) and arrive at the green square shown in the second row, and then return back to the initial position. The colored dots are the point cloud generated from the LiDAR sensing. Upper and lower 2.5 D color maps represent the value function and the elevation map, respectively. The height of the value function represents the state value. The planned expected trajectories are shown in black curves. The small axes represent the pose of the robot.

environments with rich elevation variations representing typical characteristics of uneven/rough terrains.

The first row of Figure 13 displays the navigation process over a complex terrain containing rocks, gravel, and short shrubs. To the robot, the rocks and short shrubs are impassable obstacles that are often difficult to map, and the gravel can disturb motions. The second row of the figure offers a detailed view of the robot's behavior guided by the computed value function. In this experiment, we demonstrate that the robot can effectively detect high-risk zones characterized by rocks and shrubs, even when relying solely on the geometric information generated by the elevation map. These high-risk zones are defined as the terrains with slopes larger than 45°. We apply importance sampling to cover these zones in the corresponding state space during planning. Consequently, a dense cluster of states, indicated by blue dots, primarily encompasses the high-risk zones. While the states in high-risk regions provide information about infeasibility of traversal, others in low-slope regions offer insights into varying degrees of navigability, depending on the magnitude of the slope. From the figure, we can see that the computed value function effectively integrates the environmental geometric information, resulting in a state-value surface with lower values over the high-slope areas. The low-valued portion of the surface reflects the increased costs of navigating the corresponding region. Thus, the associated policy steers the robot away from such hazardous areas. Additionally, because in this scenario the value function is computed over a relatively large area, the disturbance caused by the gravel terrain can be effectively reduced as long as the robot remains within the area. The final snapshot shows that the robot can successfully follow the policy to reach the goal region, represented by the value function's maximum, demonstrating the efficacy of our method in this representative terrain.

The third row of Figure 13 shows another environment that presents unique challenges with a small hill peaking at approximately 1m in height, a steep slope in the middle, and a mild slope situated to the left of the robot's initial position. The state space is defined over a 10 m^2 elevation map, and the states are sampled using importance sampling, where the slope determines the weights. The robot's initial position is

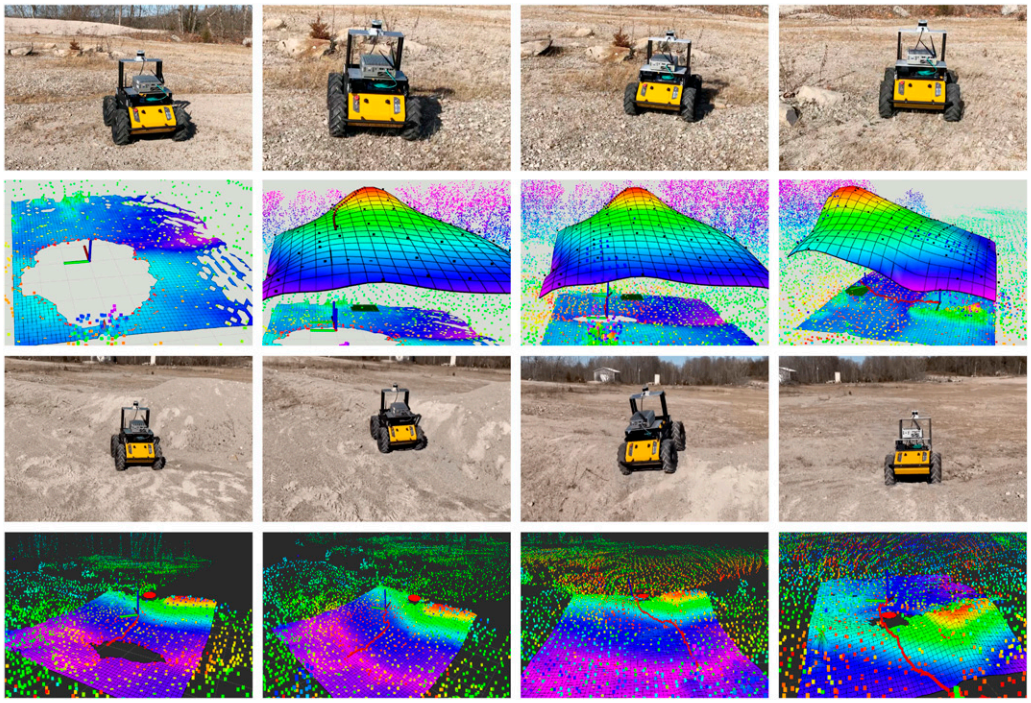

Figure 13. Snapshots of navigation in two elevation-rich environments. The first and second rows show the robot's planning in an environment scattered with rocks and the corresponding visualization for computing the solution. In each column, the visualization showcases the elevation map and the value function, represented as the lower and upper surfaces, respectively. Supporting states are marked with black dots for low-slope regions and blue dots for high-slope regions. High-slope regions present a greater navigation risk for the robot and should be avoided. The target destination for the robot is marked by a dark-green square. The red trajectory in the last column demonstrates the robot's traversed path. The last two rows demonstrate the planning over an area containing a small hill and the corresponding visualization. In the visualization, the red trajectory is the robot's traversed path and the red circle denotes the robot's planning goal. The color-coded surface represents the elevation map (for better visualization, the value function surface is not shown in this scenario).

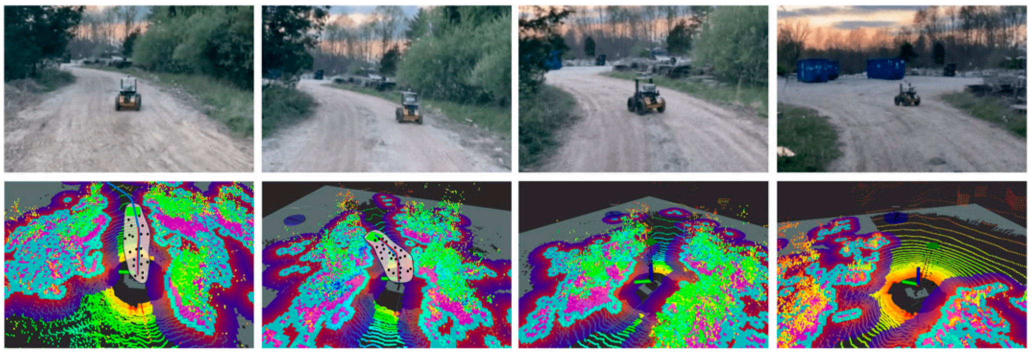

Figure 14. Snapshots of navigation in a trail using trajectory sampling to sample the states and cost map to represent the environment. The visualization in the second row shows the cost map as a $2D$ color map, where a darker color means a higher cost. In the first two visualizations, thickened/emphasized annotations are overlaid on top of the cost map. Specifically, the red and blue trajectories represent the expected path by executing the policy and the global path, respectively. The scattered black dots adjacent to the global trajectory represent a selected subset of sampled states, included specifically for illustrative purposes. Furthermore, the region encapsulating these states signifies the effective domain of the policy. This is defined as the area within which the value function yields a positive value. The goal for each time step is shown as a green square. The final goal is the blue circle.

purposefully set to face the steepest part of the terrain, directly confronting the environment's most challenging portion. We use the frontier-based method (Yamauchi 1997) to select temporary goals. As shown in Figure 13, the temporary goal, indicated as the red circle, is placed in the middle of the frontier region, located close to the obstructed area by the steepest and highest terrain segments. Such goal selection poses a challenge for the robot. Although the shortest path between the robot's initial position and the goal is a straight line across the steep terrain, the risk of flipping or tipping over is also the largest. Despite this challenge, as seen in the first three columns of Figure 13, our system effectively guides the robot to perform safe maneuvers to circumvent the steep terrain. This result proves that our method can effectively guide the robot to navigate hazardous terrains by penalizing the robot's movement distance using the first moment and incorporating the motion uncertainty via the second moment. The last column shows that after arriving at the current goal, the vehicle continues its mission by extending its elevation map, updating the MDP, and computing a new policy based on the goal selected by the frontier-based method.

8.3.2. Navigation on an unstructured construction site. In contrast to the elevated environment, Figure 14 shows the robot navigating a trail surrounded by dense vegetation around a construction site. Though lacking the challenges of steep slopes, this environment introduces new complexities. Particularly, dense vegetation makes up irregular and cluttered obstacles, and the unpaved road causes significant motion disturbance due to small rocks.

In order to adapt to this environment, we leverage a cost map to assign obstacle penalties. Due to the absence of elevation we employ the trajectory sampling mode with a global planner (ARA*). The global planner is designed for obstacle avoidance on flat surfaces. We set the temporary goal on the global path segment $6m$ ahead of the robot. Because the global planner does not account for the uncertainty introduced by the unpaved trail, following the global path without considering these uncertainties may lead to large deviations and potential failures. Our method can naturally handle this problem as shown in Figure 14. This figure demonstrates the robot's behavior at 4 timesteps. The first two columns of the second row provide visualization of the planning process. Specifically, the black dots represent the sampled supporting states around the blue global trajectory, closely followed by the expected robot's trajectory computed by our method, shown in red. In contrast to the deterministic trajectory optimization, which provides only a single trajectory, our method provides a feedback policy in a local space, represented as the half-transparent irregular shape encapsulating the supporting states. This policy region, defined by the spread of the supporting states and the second moment of the stochastic motion, enables the robot to remain within the global-path-guided region while navigating toward the goal. Thus, the system remains robust even when small rocks or other minor obstacles perturb the robot away from the globally planned path. This capability enhances its resilience to typical disturbances of unpaved paths and is critical for the safety of our robot.

9. Conclusion and discussion

This paper presents a new decision-making framework for robot planning and control in complex and unstructured environments such as the off-road navigation. We propose a method to solve the continuous-state MDP 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 validated the proposed method through thorough evaluations 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 the presented algorithm.

In addition to the theoretical contribution, our real-world experiments reveal several challenges of applying the proposed method in practice. First, the precision of the optimal solution is contingent upon the specific locations of the supporting states within the kernel representation of the value function. Identifying the most suitable locations for these states is an important aspect that impacts the accuracy of the solution. One of our future focused areas will be designing techniques to determine these optimal state locations such as leveraging more advanced kernels as well as utilizing more advanced hyperparameter learning schemes. Moreover, we aim to extend this investigation into the scalability of our approach within high-dimensional spaces. Addressing the challenges associated with scalability in the expansive spaces stands as an important objective for our forthcoming research efforts. Second, we present a system (Section 6) which is mainly responsible for converting the raw sensor data to an MDP problem by assuming that feasible/infeasible regions can be distinguished and the features in the environment (elevation in our experiment) can be obtained accurately from sensors. Since this MDP is used for computing a policy, ensuring the MDP matches the real-world scenario is paramount in the final performance of the system. However, the map built from the noisy sensor measurement usually cannot accurately represent the geometry of the environment, for example, occupancy and elevation. The MDP derived from this inaccurate map may deviate from the real environment. Thus, reasoning about these inaccuracies in the planning method is essential for building a robust navigation system. Additionally, an accurate physics model for planning is necessary in complex environments, where detailed motion control strategies are needed. By comparing the results of Sections 5.1 and 8, we can observe that the planning method can generate more efficient policies if we use a physics model which can better describe the robot's motion. In our real-world experiments, our modeling of the first moment uses inaccurate models to describe the physical interaction between the robot and the terrain. Although these models can be used for planning in the particular environments we tested, to generalize to more complex situations, it is necessary to develop more accurate physical models that consider not only the elevation but also terrain textures, which is our future work.

Acknowledgments

The authors would like to express their sincere gratitude to all anonymous reviewers for their invaluable and constructive feedback, which significantly contributed to the improvement and quality enhancement of this manuscript.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The authors disclosed receipt of the following financial support for the research, authorship, and/or publication of this article. This work has been supported by the Army Research Office and was accomplished under Cooperative Agreement Numbers W911NF-20-2-0099 and W911NF-22-2-0018: Scalable, Adaptive, and Resilient Autonomy (SARA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. This work was also partially supported by NSF: RI: Small: Exploiting Symmetries of Decision Theoretic Planning for Autonomous Vehicles (grant no 2006886), and NSF: CAREER: Autonomous Live Sketching of Dynamic Environments by Exploiting Spatiotemporal Variations (grant no 2047169).

ORCID iDs

Junhong Xu https://orcid.org/0000-0001-7127-5093
Jason M Gregory https://orcid.org/0000-0002-3929-6422
Lantao Liu https://orcid.org/0000-0002-6796-6817

References

Agha-Mohammadi A-A, Chakravorty S and Amato NM (2014) Firm: sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. *The International Journal of Robotics Research* 33(2): 268–304.

Al-Sabban H, Gonzalez LF and Smith RN (2013) Wind-energy based path planning for unmanned aerial vehicles using Markov decision processes. In 2013 IEEE International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany, 6–10 May 2013.

Allgöwer F and Zheng A (2012) *Nonlinear Model Predictive Control*. Basel: Birkhäuser.

Althoff M, Stursberg O and Buss M (2008) Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In 2008 47th IEEE Conference on Decision and Control, Cancun, Mexico, 9–11 December 2008.

Antos A, Szepesvári C and Munos R (2008) Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning* 71(1): 89–129.

Arulkumaran K, Deisenroth MP, Brundage M, et al. (2017) *A Brief Survey of Deep Reinforcement Learning*. arXiv preprint arXiv:1708.05866.

Baek SS, Kwon H, Yoder JA, et al. (2013) 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, Tokyo, Japan, 03–07 November 2013.

Bansal S, Chen M, Herbert S, et al. Hamilton-Jacobi reachability: a brief overview and recent advances. In 2017 IEEE 56th Annual Conference on Decision and Control (CDC), Melbourne, VIC, Australia, 12–15 December 2017.

Bellman RE (2015) *Adaptive Control Processes: A Guided Tour*. Princeton: Princeton university press.

Bemporad A and Morari M (1999) Robust model predictive control: a survey. *Robustness in Identification and Control*. Berlin, Germany: Springer.

Bertsekas DP (2005) Dynamic programming and suboptimal control: a survey from adp to mpc. *European Journal of Control* 11(4-5): 310–334.

Bertsekas DP (2011) Approximate policy iteration: a survey and some new methods. *Journal of Control Theory and Applications* 9(3): 310–335.

Bertsekas D (2012) *Dynamic Programming and Optimal Control*. Nashua, NH: Athena scientific.

Bertsekas DP and Tsitsiklis JN (1996) *Neuro-Dynamic Programming*. Nashua, NH: Athena Scientific Belmont.

Boutilier C, Dean T and Hanks S (1999) Decision-theoretic planning: structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11(1–94): 1–94.

Braverman A, Gurvich I and Huang J (2020) On the taylor expansion of value functions. *Operations Research* 68(2): 631–654.

Bresson G, Alsayed Z, Yu L, et al. (2017) Simultaneous localization and mapping: a survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles* 2(3): 194–220.

Burgard W, Moors M, Stachniss C, et al. (2005) Coordinated multi-robot exploration. *IEEE Transactions on Robotics* 21(3): 376–386.

Chen J, Su K and Shen S (2015) Real-time safe trajectory generation for quadrotor flight in cluttered environments. In 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO), Zhuhai, China, 06–09 December 2015.

Das MP, Conover DM, Eum S, et al. (2022) MA3: model-accuracy aware anytime planning with simulation verification for navigating complex terrains. *Proceedings of the International Symposium on Combinatorial Search* 15(1): 65–73.

Deisenroth MP, Rasmussen CE and Peters J (2009) Gaussian process dynamic programming. *Neurocomputing* 72(7-9): 1508–1524.

Deits R and Tedrake R (2015) Efficient mixed-integer planning for uavs in cluttered environments. In 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015.

Elfwing S, Uchibe E and Doya K (2018) Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks : The Official Journal of the International Neural Network Society* 107(3–11): 3–11.

Engel Y, Mannor S and Meir R (2003) Bayes meets bellman: the Gaussian process approach to temporal difference learning. Proceedings of the 20th International Conference on Machine Learning, Washington, DC, USA, 21–24 August 2003.

Evans LC (2010) Partial differential equations. *Graduate Series in Mathematics*. 2nd edition. Providence, RI: American Mathematical Society.

Fankhauser P, Bloesch M and Hutter M (2018) Probabilistic terrain mapping for mobile robots with uncertain localization. *IEEE Robotics and Automation Letters* 3(4): 3019–3026.

Fu Y, Xiang Y and Zhang Y (2015) Sense and collision avoidance of unmanned aerial vehicles using markov decision process and flatness approach. In 2015 IEEE International Conference on Information and Automation, Lijiang, China, 08–10 August 2015.

Gammell JD and Strub MP (2021) Asymptotically optimal sampling-based motion planning methods. *Annual Review of Control, Robotics, and Autonomous Systems* 4: 295–318.

Gao F and Shen S (2016) Online quadrotor trajectory generation and autonomous navigation on point clouds. In 2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Lausanne, Switzerland, 23–27 October 2016.

Gao F, Wu W, Lin Y, et al. (2018) Online safe trajectory generation for quadrotors using fast marching method and bernstein basis polynomial. In 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 21–25 May 2018.

Gao F, Wu W, Gao W, et al. (2019) Flying on point clouds: online trajectory generation and autonomous navigation for quadrotors in cluttered environments. *Journal of Field Robotics* 36(4): 710–733.

Gordon G (1999) *Approximate Solutions to Markov Decision Processes*. Pittsburgh, PA: Carnegie-Mellon University School of Computer Science. Technical report.

Gorodetsky AA, Karaman S and Marzouk YM (2015) Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition. https://ilp.mit.edu/node/42974

Gregory J, Fink J, Stump E, et al. (2016) Application of multi-robot systems to disaster-relief scenarios with limited communication. *Field and Service Robotics*. Berlin, Germany: Springer, 639–653.

Grisetti G, Kümmerle R, Stachniss C, et al. (2010) A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine* 2(4): 31–43.

Heess N, Wayne G, Silver D, et al. (2015) Learning continuous control policies by stochastic value gradients. *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 2944–2952.

Hofmann T, Schölkopf B and AlexanderSmola J (2008) *Kernel Methods in Machine Learning*. Cambridge, MA: The Annals of Statistics, 1171–1220.

Howard TM and Kelly A (2007) Optimal rough terrain trajectory generation for wheeled mobile robots. *The International Journal of Robotics Research* 26(2): 141–166.

Huynh A, Karaman S and Frazzoli E (2016) An incremental sampling-based algorithm for stochastic optimal control. *The International Journal of Robotics Research* 35(4): 305–333.

Islam R, Henderson P, Gomrokchi M, et al. (2017) *Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control*. arXiv preprint arXiv: 1708.04133.

Johnson SG The NLopt nonlinear-optimization package. https://ab-initio.mit.edu/nlopt

Kalman RE (1960) Contributions to the theory of optimal control. *Bol. soc. mat. mexicana* 5(2): 102–119.

Kappen H, Gómez V and Opper M (2012) Optimal control as a graphical model inference problem. *Machine Learning* 87(2): 159–182.

Karaman S and Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research* 30(7): 846–894.

Kaufmann E, Leonard B, Loquercio A, et al. (2023) *Champion-Level Drone Racing Using Deep Reinforcement Learning*. London, UK: Nature Publishing Group UK.

Kober J, Bagnell JA and Peters J (2013) Reinforcement learning in robotics: a survey. *The International Journal of Robotics Research* 32(11): 1238–1274.

Kuss M and Rasmussen CE (2004) Gaussian processes in reinforcement learning. *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 751–758.

Lagoudakis MG and Parr R (2003) Least-squares policy iteration. *Journal of Machine Learning Research* 4(Dec): 1107–1149.

Langson W, Chryssochoos I, Raković SV, et al. (2004) Robust model predictive control using tubes. *Automatica* 40(1): 125–133.

LaValle SM (2006) *Planning Algorithms*. Cambridge, MA: Cambridge University Press.

Likhachev M, Ferguson DI, J Gordon G, et al. (2005) Anytime dynamic a*: an anytime, replanning algorithm. *ICAPS* 5: 262–271.

Lillicrap TP, Hunt JJ, Alexander P, et al. (2015) *Continuous Control With Deep Reinforcement Learning*. arXiv preprint arXiv:1509.02971.

Liu L and Sukhatme S (2018) A solution to time-varying Markov decision processes. *IEEE Robotics and Automation Letters* 3(3): 1631–1638.

Liu S, Watterson M, Mohta K, et al. (2017) Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. *IEEE Robotics and Automation Letters* 2(3): 1688–1695.

Lutter M, Mannor S, Peters J, et al. (2021) *Value Iteration in Continuous Actions, States and Time*. arXiv preprint arXiv: 2105.04682.

Majumdar A and Tedrake R (2013) Robust online motion planning with regions of finite time invariance. *Algorithmic Foundations of Robotics X*. Berlin, Germany: Springer, 543–558.

Majumdar A and Tedrake R (2017) Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research* 36(8): 947–982.

Makoviychuk V, Wawrzyniak L, Guo Y, et al. (2021) *Isaac Gym: High Performance Gpu-Based Physics Simulation for Robot Learning*. arXiv preprint arXiv:2108.10470.

Martin J, Wang J and Englot B (2018) *Sparse Gaussian Process Temporal Difference Learning for Marine Robot Navigation*. arXiv preprint arXiv:1810.01217.

Maurette M (2003) Mars rover autonomous navigation. *Autonomous Robots* 14(2-3): 199–208.

McEwen A, Eliason EM, Bergstrom JW, et al. (2007) Mars reconnaissance orbiter's high resolution imaging science experiment (hirise). *Journal of Geophysical Research: Planets* 112(E5).

Mellinger D and Kumar V (2011) Minimum snap trajectory generation and control for quadrotors. In 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 09–13 May 2011.

Miki T, Lee J, Hwangbo J, et al. (2022) Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics* 7(62): eabk2822.

Munos R and Moore A (2002) Variable resolution discretization in optimal control. *Machine Learning* 49(2-3): 291–323.

Munos R and Szepesvári C (2008) Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research* 9(May): 815–857.

Nair A, McGrew B, Andrychowicz M, et al. (2018a) Overcoming exploration in reinforcement learning with demonstrations. 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018.

Nair V, Vitchyr P, Dalal M, et al. (2018b) Visual reinforcement learning with imagined goals. *Advances in Neural Information Processing Systems* 31.

Oleynikova H, Burri M, Taylor Z, et al. (2016) Continuous-time trajectory optimization for online uav replanning. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea, 09–14 October 2016.

Otte M, Silva W and Frew E (2016) Any-time path-planning: time-varying wind field+ moving obstacles. In 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 16–21 May 2016.

Pereira AA, Binney J, Hollinger GA, et al. (2013) Risk-aware path planning for autonomous underwater vehicles using predictive ocean models. *Journal of Field Robotics* 30(5): 741–762.

Powell WB (2016) Perspectives of approximate dynamic programming. *Annals of Operations Research* 241(1-2): 319–356.

Puterman ML (2014) *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ: John Wiley & Sons.

Rawlings JB, Mayne DQ and Diehl M (2017) *Model Predictive Control: Theory, Computation, and Design*. Madison, WI: Nob Hill Publishing Madison.

Richter C, Adam B and Roy N (2016) Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. *Robotics Research*. Berlin, Germany: Springer, 649–666.

Schulman J, Filip W, Dhariwal P, et al. (2017) *Proximal Policy Optimization Algorithms*. arXiv preprint arXiv: 1707.06347.

Shawe-Taylor J and Cristianini N (2004) *Kernel Methods for Pattern Analysis*. Cambridge, MA: Cambridge University Press.

Si J, Barto AG, Powell WB, et al. (2004) *Handbook of Learning and Approximate Dynamic Programming*. Hoboken, NJ: John Wiley & Sons.

Sun S, Romero A, Foehn P, et al. (2022) A comparative study of nonlinear mpc and differential-flatness-based control for quadrotor agile flight. *IEEE Transactions on Robotics* 38(6): 3357–3373.

Sutton RS and Barto AG (2018) *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT press.

Taylor G and Parr R (2009) Kernelized value function approximation for reinforcement learning. Proceedings of the 26th Annual International Conference on Machine Learning, Montreal, Canada, 14 June 2009.

Tedrake R, Manchester IR, Tobenkin M, et al. (2010) Lqr-trees: feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research* 29(8): 1038–1052.

Theodorou E, Jonas B and Schaal S (2010) A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research* 11: 3137–3181.

Thrun S, Burgard W and Fox D (2000) *Probabilistic robotics*. Cambridge, MA: MIT press Cambridge.

Van Den Berg J, Abbeel P and Goldberg K (2011) Lqg-mp: optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research* 30(7): 895–913.

Wang J, Triest S, Wang W, et al. (2021) Rough terrain navigation using divergence constrained model-based reinforcement learning. 5th Annual Conference on Robot Learning, London, UK, 8–11 November 2021.

Webb DJ and van den Berg J (2012) *Kinodynamic Rrt*: Optimal Motion Planning for Systems With Linear Differential Constraints*. arXiv preprint arXiv:1205.5088.

Williams G, Nolan W, Goldfain B, et al. (2017) Information theoretic mpc for model-based reinforcement learning.

In 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May 2017.

Williams G, Goldfain B, Paul D, et al. (2018) Robust sampling based model predictive control with sparse objective information. *Robotics Science and Systems*. Pittsburgh, PA: Carnegie Mellon University.

Xu X, Hu D and Lu X (2007) Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks* 18(4): 973–992.

Xu J, Yin K and Liu L (2019) Reachable space characterization of Markov decision processes with time variability. *Proceedings of Robotics: Science and Systems*. Germany: Freiburgim-Breisgau. DOI: 10.15607/RSS.2019.XV.069.

Xu J, Yin K and Liu L (2020) Kernel taylor-based value function approximation for continuous-state Markov decision processes. *Proceedings of Robotics: Science and Systems*. Corvalis, Oregon, USA: Carnegie Mellon University. DOI: 10.15607/rss.2020.xvi.050.

Yamauchi B (1997) A frontier-based approach for autonomous exploration. Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIR-A'97.'Towards New Computational Principles for Robotics and Automation', Monterey, CA, USA, 10–11 July 1997.

Zhou B, Gao F, Wang L, et al. (2019) Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters* 4(4): 3529–3536.

Zhou B, Gao F, Pan J, et al. (2020) Robust real-time uav re-planning using guided gradient-based optimization and topological paths. In 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 31 May 2020.

Appendix

Table 2. A index to multimedia extensions.

Extension	Media type	Description
1	Video	Indoor experiment video
2	Video	Outdoor experiment video showing the vehicle can traverse various types of terrains
3	Video	Simulation experiment video comparing our method and NMPC