Latent Belief Space Motion Planning under Cost, Dynamics, and Intent Uncertainty

Dicong Qiu
isee.ai
Cambridge, MA 02139
Email: dq@isee.ai

Yibiao Zhao
isee.ai
Cambridge, MA 02139
Email: yz@isee.ai

Chris L. Baker
isee.ai
Cambridge, MA 02139
Email: chrisbaker@isee.ai

***Abstract*—Autonomous agents are limited in their ability to observe the world state. Partially observable Markov decision processes (POMDPs) model planning under world state uncertainty, but POMDPs with multimodal beliefs, continuous actions, and nonlinear dynamics suitable for robotics applications are challenging to solve. We present a dynamic programming algorithm for planning in the belief space over discrete latent states in POMDPs with continuous states, actions, observations, and nonlinear dynamics. Unlike prior belief space motion planning approaches which assume unimodal Gaussian uncertainty, our approach constructs a novel tree-structured representation of possible observations and multimodal belief space trajectories, and optimizes a contingency plan over this structure. We apply our method to problems with uncertainty over the reward or cost function (e.g., the configuration of goals or obstacles), uncertainty over the dynamics, and uncertainty about interactions, where other agents' behavior is conditioned on latent intentions. Three experiments show that our algorithm outperforms strong baselines for planning under uncertainty, and results from an autonomous lane changing task demonstrate that our algorithm can synthesize robust interactive trajectories.**

I. Introduction

Planning under uncertainty resulting from a limited ability to observe the world state is a critical capacity for autonomous agents. Noisy actuators, imperfect sensors, and perceptual limitations such as occlusion contribute to the uncertainty that agents face when deciding to act. Even with perfect sensors and perception, latent states of the world can remain opaque, such as whether there is an open parking spot on the next block, or whether another driver intends to yield. Planning under this uncertainty requires balancing the cost of exploratory actions with the potential benefit of the information gained. However, the problem of planning under partial observability, which can be formalized as a partially observable Markov decision process (POMDPs), is generally intractable [1].

Trajectory optimization techniques have proven very effective for robotics applications [2, 3], but typically require the state to be fully observable (or separately estimated). Extensions of trajectory optimization to belief space planning allow partial observability to be captured within motion planning algorithms suitable for continuous, nonlinear robotic systems [4, 5]. However, these approaches primarily consider unimodal Gaussian uncertainty, which limits their utility for problems with multimodal structure. Real-world uncertainty is multimodal: for example, in autonomous driving there is uncertainty about the presence and state of vehicles that are occluded or out of sensor range, about the functional mode of the vehicle and its components, about the state of the environment, and about the characteristics, intentions and beliefs of other drivers. This multimodal structure can be represented within general POMDPs, which can be solved by general-purpose solvers [6, 7]. However, optimizing continuous actions for motion planning is challenging for state of the art POMDP solvers.

In this paper, we propose a trajectory optimization approach for solving nonlinear, continuous state, action, and observation POMDPs with non-Gaussian beliefs over discrete latent variables. Our approach, called Partially Observable Differential Dynamic Programming (PODDP), builds and optimizes a contingency plan over a tree of possible observations and trajectories in the belief space. We derive techniques for dynamic programming over the trajectory tree, which involve propagating an approximate value function through the belief state dynamics defined by observations and Bayesian belief updating. Lastly, we describe a practical hierarchical dynamic programming decomposition of the problem, which greatly enhances the efficiency of our approach.

PODDP solves several important classes of nonlinear, continuous planning problems with uncertainty over discrete latent states: (1) Tasks where the cost function depends on an uncertain latent state, e.g., where an agent must approach or avoid goals or obstacles which may be in a finite number of locations. (2) Tasks where the dynamics are conditioned on the uncertain latent mode of the system, e.g., contact mode, component status, or environmental condition (e.g., smooth vs. rough terrain). (3) Interactive tasks where other agents' trajectories impose dynamic costs and are influenced by their latent intentions, e.g., autonomous driving systems must plan under uncertainty about other vehicles' interactive trajectories, conditioned on their drivers' situational awareness level, intention to cooperate, etc.

We demonstrate the efficacy of our method experimentally in instances of each of these problem classes. We compare our algorithm with two strong baselines: 1) a "fully observable" baseline under which DDP treats the maximum-likelihood state as the true world state, and 2) a baseline which combines multiple latent state hypotheses inside the cost function, weighted by their probabilities. Lastly, we consider

a challenging autonomous driving setting in which the model must plan interactive lane changing trajectories, and show that PODDP can plan and execute successful lane change trajectories by inferring whether another agent intends to yield.

II. Related work

In this section we first review related research on planning under partial observability, with a focus on methods for planning in continuous action spaces. We review techniques for Gaussian belief space planning, as well as general-purpose POMDP solvers which can plan continuous actions. Finally, we review research applying related techniques, primarily in applications related to autonomous driving.

A. Gaussian belief space planning

Trajectory optimization methods compute a sequence of continuous actions to minimize the expected cost of the resulting state sequence. For the special case of problems with linear dynamics, quadratic costs, and additive Gaussian process and observation noise, optimal continuous feedback policies can be computed by the LQG algorithm using dynamic programming [8]. In the LQG setting, the separation principle [9] implies that the same feedback policy is optimal when applied to either the true state in the fully observable setting, or to the estimated state in the partially observable setting. In the nonlinear setting, differential dynamic programming (DDP) [10] and iterative LQG (iLQG) [11, 3] extend the dynamic programming approach to compute locally optimal feedback policies for systems with smooth, nonlinear dynamics and non-quadratic costs. Like LQG, these algorithms separate estimation and control, and thus they are unable to plan exploratory actions, because they do not explicitly model the effect of observations on the belief dynamics.

Methods for trajectory optimization in Gaussian belief space [12, 4, 13, 5] model the belief dynamics induced by Gaussian process and observation noise by augmenting the state to include the estimated mean and covariance, and propagating the belief state with a Kalman filter. These methods use two main ways to approximate the belief dynamics, both of which yield policies which explore the environment to gain information that is useful for optimizing future value. The first approximation assumes that the observations from each state take their maximum-likelihood values [12, 13, 5], which makes the belief space dynamics deterministic. This heuristic reduces to the LQG algorithm in the linear-quadratic-Gaussian setting, and provides a lower bound on the uncertainty of future belief states in the nonlinear setting. The second approximation propagates the belief state by propagating the linearized Gaussian belief state dynamics within each step of dynamic programming. [4] showed that this approach outperforms the maximum-likelihood observations assumption in a Gaussian belief space planning task. However, because this approximation is only applicable in the Gaussian case, our approach in this paper is more analogous to the maximum-likelihood observations approximation.

Other techniques use LQG controllers to facilitate sampling-based Gaussian belief space motion planning. [14] and [15] use LQG and Kalman filtering to estimate the expected covariance, cost, and success probability of tracking trajectories computed by RRT in the presence of motion and sensor noise. The results of both methods demonstrate that they are able to plan information gathering, exploratory actions. The FIRM algorithm [16] constructs an information-state roadmap, an extension of probabilistic roadmaps [17] to problems with motion and sensing uncertainty, where nodes and edges lie in the belief space, and uses LQG controllers to break the curse of history and guarantee reachability of nodes in the roadmap.

There are several reinforcement learning techniques (RL) for solving continuous state-action-observation POMDPs. The model-based approach of [18] extends the PILCO algorithm to continuous state-action Gaussian POMDPs, by simultaneously learning a Gaussian process dynamical model and policy. [19] uses model-free deep variational RL to jointly learn a dynamical model and a continuous policy, and applies this method to a Gaussian mountain hike problem.

Gaussian belief space planning assumes that all uncertainty can be represented in the form of unimodal Gaussian distributions over the state space. In contrast, our work here captures the multimodality of real-world uncertainty. In a similar spirit to our work, [20] proposes an extension of the belief space planning method of [12] to hybrid continuous dynamics, in which a (partially observable) discrete mode determines the continuous dynamics of the system. Several other techniques have been developed for this setting, including the Sequential Action Control approach of [21], and the point-based method of [22] (which assumes discrete actions).

B. General purpose POMDP solvers

Point-based solvers [6] allow medium-sized POMDPs to be solved offline, and have been extended to continuous state spaces [23], but require discretization of the action and observation spaces. Monte-Carlo tree search (MCTS) is an online method which can scale to large domains [24, 7]. MCTS can naturally handle continuous state spaces, but continuous observations and actions are more complicated. Fine discretization of the observation space is feasible because MCTS can scale to large observation spaces; however, MCTS scales exponentially in the number of actions, so only coarse discretization of actions is tractable [25]. Several techniques have been proposed to handle continuous actions and observations. The approach of [26] applies double progressive widening [27] to actions and observations in MCTS, while [28] performs local search in the belief tree for optimal continuous actions.

C. Related applications

The active SLAM problem is similar in spirit to the problem classes we consider involving uncertainty about the spatial structure of the environment (see Experiments 1 and 2). The large literature in this area [29] is outside the scope of this review. [30] presents a belief space planning approach to

the active SLAM problem, and augments Gaussian dynamics and observations with binary random variables representing whether a measurement is taken. [5] scales belief space planning to 50 landmarks in an active SLAM experiment.

The large literature on interactive and intention-aware planning approaches for mobile robots is highly relevant to our work. Although planning for interaction using a world model that includes (a distribution over) other agents' predicted trajectories as part of the dynamics – treating other agents as "part of the environment" – is a popular technique [31, 32, 33, 34, 35], planning over uncertain predictions of other agents' trajectories can lead to the freezing robot problem [36], while treating other agents' trajectories as deterministically predictable can lead to overly aggressive behavior [32, 35]. A variety of solutions have been proposed. Intention-aware POMDP formulations of interaction are conceptually similar to the approach we take: they solve a POMDP with a belief state defined by hidden intent variables which influence the predicted trajectory for each agent [31, 26, 34]. [37] considers interactive behavior that is governed by a continuous latent variable which determines other drivers' "social value orientation" (SVO). The algorithm computes a game-theoretic equilibrium, given SVO estimates, for all vehicles simultaneously using multi-agent trajectory optimization techniques. We take inspiration from the approach of trajectory-optimization through differentiable models of other agents in [32]. Although we consider a simpler class of agent models than the ones in that paper, our approach can be applied to richer agent models, and it has the advantage of handling uncertainty about agents' intentions. [38] considers conceptually similar autonomous driving tasks benefiting from information gathering actions to infer the mental state of another driver (attentive or distracted); however, unlike our work, their approach does not optimize the tradeoff between exploration and exploitation. The algorithm we propose is structurally similar to the contingency-planning approach of [33]; however, our method operates over observation horizons longer than one step, and takes a DDP-based approach to trajectory optimization.

III. Problem formulation

We consider discrete-time, finite-horizon POMDPs with hybrid continuous and discrete states, continuous actions, and continuous observations. The discrete state is dynamic and partially observable, and the belief over this state can represent multimodal, time-varying uncertainty about the robot state, the world state, or the state of other agents. For simplicity we assume the continuous state is fully observable, and only the discrete state is partially observable; our formulation is thus a mixed-observability MDP model [39], which yields a compact representation of the belief space and dynamics. We partition the state space $\mathcal{S} = \mathcal{X} \times \mathcal{Z}$ into a continuous state space $\mathcal{X}$, and a finite latent state space $\mathcal{Z}$. We denote the continuous control space $\mathcal{U}$, and the continuous observation space $\mathcal{O}$.

The system dynamics are defined by the conditional distribution over the next state $\langle \mathbf{x}_{t+1}, z_{t+1} \rangle \in \mathcal{S}$, $p(\mathbf{x}_{t+1}, z_{t+1} | \mathbf{x}_t, z_t, \mathbf{u}_t)$, which depends on the current state $\langle \mathbf{x}_t, z_t \rangle \in \mathcal{S}$, and control $\mathbf{u}_t \in \mathcal{U}$. The distribution over observations $\mathbf{o}_t \in \mathcal{O}$, $p(\mathbf{o}_t | \mathbf{x}_t, z_t)$ is also conditioned on the current state. The graphical model shown in Fig. 1 defines the conditional dependencies in the model.

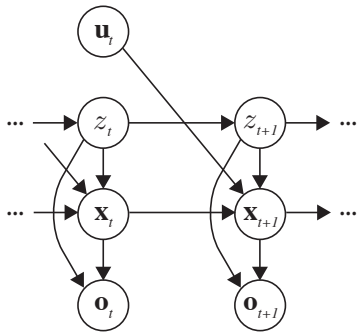

Fig. 1: Graphical model of our POMDP formulation. The continuous state $\mathbf{x}_t$ and observation $\mathbf{o}_t$ are observed variables, $\mathbf{u}_t$ is the control, and z_t is the dynamic partially observable discrete latent state.

The belief about the dynamic latent state z_t depends on the history of observed states, controls, and observations. We use recursive Bayesian filtering over the conditional dependency structure in Fig. 1 to update the belief, given the latest observation, which (based on the mixed-observability assumption) includes both $\mathbf{o}_{t+1}$ and $\mathbf{x}_{t+1}$:

$$\begin{aligned}\mathbf{b}_{t+1}(z_{t+1}) &\triangleq P(z_{t+1} | \mathbf{o}_{t+1}, \mathbf{x}_{t+1}, \mathbf{u}_t, \mathbf{x}_t, \ldots, \mathbf{o}_1, \mathbf{x}_1, \mathbf{u}_0, \mathbf{x}_0, \mathbf{b}_0) \\ &= P(z_{t+1} | \mathbf{o}_{t+1}, \mathbf{x}_{t+1}, \mathbf{x}_t, \mathbf{u}_t, \mathbf{b}_t) \\ &= \eta \cdot p(\mathbf{o}_{t+1} | \mathbf{x}_{t+1}, z_{t+1}) p(\mathbf{x}_{t+1} | z_{t+1}, \mathbf{x}_t, \mathbf{u}_t) \\ &\quad \cdot \sum_{z_t \in \mathcal{Z}} P(z_{t+1} | z_t) \mathbf{b}_t(z),\end{aligned} \tag{1}$$

where η is a normalizing constant. We define the belief update function $h(\mathbf{o}_{t+1}, \mathbf{x}_{t+1}, \mathbf{x}_t, \mathbf{u}_t, \mathbf{b}_t)$ to denote the deterministic belief dynamics, such that the mapping from $\mathbf{b}_t$ to $\mathbf{b}_{t+1}$ satisfies (1).

The running loss function $l(\mathbf{x}_t, z_t, \mathbf{u}_t)$ represents the cost incurred by the current state and control, and the final loss function $l_f(\mathbf{x}_T, z_T)$ represents the cost at the end of the planning horizon; both functions are assumed to have positive semi-definite Hessian matrices. For compactness, we also define the belief state loss function as the expected cost under the current belief:

$$\begin{aligned} l(\mathbf{x}_t, \mathbf{b}_t, \mathbf{u}_t) &= \mathbb{E}_{z_t \sim \mathbf{b}_t} l(\mathbf{x}_t, z_t, \mathbf{u}_t) \\ l_f(\mathbf{x}_T, \mathbf{b}_T) &= \mathbb{E}_{z_T \sim \mathbf{b}_T} l_f(\mathbf{x}_T, z_T). \end{aligned} \tag{2}$$

The objective to be minimized is the expected cumulative cost-to-go, which is a function of the current belief state and a closed-loop policy $\pi_t(\mathbf{x}_t, \mathbf{b}_t)$ mapping belief states to controls:

$$J_t^\pi(\mathbf{x}_t, \mathbf{b}_t) = \mathop{\mathbb{E}}_{\mathbf{o}_{t+1:T}} \left[\sum_{\tau=t}^{T-1} l(\mathbf{x}_\tau, \mathbf{b}_\tau, \pi_\tau(\mathbf{x}_\tau, \mathbf{b}_\tau)) + l_f(\mathbf{x}_T, \mathbf{b}_T) \right]. \tag{3}$$

The value function maps each belief state to the cost-to-go of the optimal policy from that state:

$$V_t(\mathbf{x}_t, \mathbf{b}_t) = \min_{\pi} J_t^{\pi}(\mathbf{x}_t, \mathbf{b}_t). \tag{4}$$

Applying the dynamic programming principle [8], we compute the value function recursively:

$$\begin{aligned} V_t(\mathbf{x}_t, \mathbf{b}_t) &= \min_{\mathbf{u}_t} \big[l(\mathbf{x}_t, \mathbf{b}_t, \mathbf{u}_t) \\ &\quad + \mathbb{E}_{\mathbf{o}_{t+1}, \mathbf{x}_{t+1}} [V_{t+1}(\mathbf{x}_{t+1}, h(\mathbf{o}_{t+1}, \mathbf{x}_{t+1}, \mathbf{x}_t, \mathbf{u}_t, \mathbf{b}_t))] \big] \\ V_T^{\pi}(\mathbf{x}_T, \mathbf{b}_T) &= l_f(\mathbf{x}_T, \mathbf{b}_T), \end{aligned} \tag{5}$$

where the expectation is over $\mathbf{o}_{t+1}, \mathbf{x}_{t+1} \sim p(\mathbf{o}_{t+1}, \mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{b}_t, \mathbf{u}_t)$, and the value at the planning horizon T is the final belief state cost. In the next section we describe our PODDP method for computing the value function and optimal policy.

IV. Partially observable differential dynamic programming

Standard dynamic programming-based trajectory optimization methods, such as DDP and iLQG, optimize a trajectory by alternating a forward pass which rolls out the dynamics and costs using a control sequence, and a backward pass which takes a local second-order approximation to the value function, and updates the control sequence to optimize this approximate value function. This process repeats until convergence to a locally optimal trajectory. Dynamic programming algorithms for Gaussian belief space planning [4] proceed similarly – they roll out a trajectory in the forward pass by propagating a belief state defined by the mean and covariance of the state, and optimize an approximate value function around the belief state trajectory in the backward pass.

PODDP plans in belief space, but unlike Gaussian belief space planning, the marginal distribution over observations is not unimodal, and the belief-space trajectory cannot be approximated by propagating a single sequence of means and variances. In our setting, the discrete latent variable z_t induces a multimodal distribution over observations, and a non-Gaussian belief state, which induces a(n infinitely branching) tree of observations, beliefs, and controls in the forward pass. The PODDP forward pass approximates this tree with a finite representation which we call a "trajectory tree", shown in Fig. 2, analogous to the policy tree representation in the classic discrete POMDP setting [40], or to the belief tree in MCTS methods [7]. The PODDP backward pass proceeds from the leaves of the tree, and propagates the value through observations and belief updates via dynamic programming. The remainder of this section will derive the forward and backward passes of the PODDP algorithm, and then derive an efficient hierarchical decomposition of the trajectory tree.

A. PODDP forward pass

Given an initial belief state $\langle \mathbf{x}_0, \mathbf{b}_0 \rangle$, the PODDP forward pass constructs a trajectory tree which approximates the infinite space of possible control, state, observation, and belief

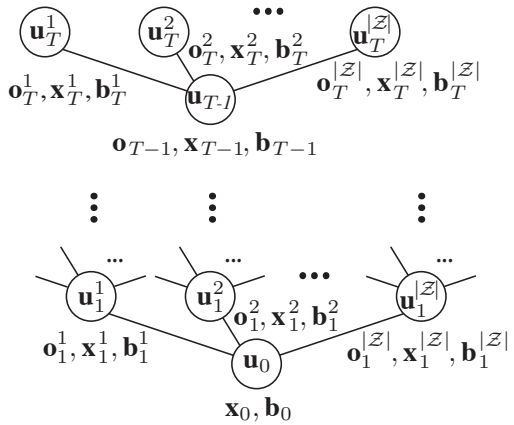

Fig. 2: PODDP trajectory tree. Starting from belief state $\langle \mathbf{x}_0, \mathbf{b}_0 \rangle$, tree construction rolls out control $\mathbf{u}_0$ for each possible latent state value $i \in \{1, \ldots, |\mathcal{Z}|\}$, assuming next state $\mathbf{x}_1^i$ and observation $\mathbf{o}_1^i$ take their maximum likelihood values, and $\mathbf{b}_1^i$ is given by Bayesian belief updating. Tree construction proceeds recursively from each $\mathbf{x}_1^i, \mathbf{b}_1^i$ until the finite horizon is reached. Note that in the last layer we have suppressed the superscript labels for $\mathbf{o}_{T-1}, \mathbf{x}_{T-1}, \mathbf{b}_{T-1}$ for clarity – the notation is cumbersome, and should record the complete history of latent state values used to generate the state and observation sequence preceding the node.

sequences up to a finite horizon T. Each node in the tree is labeled by the control to be executed if that node is reached. From each node, a finite set of branches is generated corresponding to possible state transitions, observations, and belief updates given the control and belief state at that node. A control node is created following each branch, and tree expansion proceeds recursively until the finite horizon is reached. Fig. 2 sketches the trajectory tree structure, and Algorithm 1 (included in online Supplementary Materials[1]) defines the algorithm formally.

To approximate the infinite set of continuous observations that are possible from each node, we introduce a maximum-likelihood outcomes (MLO) assumption [12]: For each possible latent state value $z \in \mathcal{Z}$, we compute the maximum likelihood state transition and observation, and perform the belief update as defined in lines 12-14 of Algorithm 1. The MLO assumption transforms the operation of sampling next states and observations into a deterministic function, which PODDP requires to be differentiable. To enable this, we assume that $p(\mathbf{x}_{t+1} | z_{t+1}, \mathbf{x}_t, \mathbf{u}_t)$ and $p(\mathbf{o}_{t+1} | \mathbf{x}_{t+1}, z_{t+1})$ are Gaussian distributions with additive noise; MLO corresponds to taking the mean of the distributions, and differentiation involves taking the derivative of the underlying process.

The forward pass is called on every iteration of PODDP. On the first iteration, the nominal controls U_{nom} are initialized to a default value (constant in our examples in this paper; more complex schemes are possible), and nominal belief states S_{nom} and control updates k and K are set to null. At later iterations, k and K, computed by the backward pass, specify modifications to the previous control U_{nom}, and provide linear

[1] Available at https://davidqiu1993.github.io/poddp-paper

feedback control gains to stabilize the trajectory around S_{nom}, respectively. The step size α of the trajectory update is set by a line search procedure [3].

B. PODDP backward pass

The PODDP backward pass operates over a trajectory tree, proceeding from the leaves, and propagating a local second-order approximation to the value function through the observations, dynamics, and belief updates that take place at each node. Locally optimal control modifications and linear feedback control gains are computed at each node, which are used to update the trajectory during the next forward pass. Algorithm 2 (see Supplementary) defines this procedure, which traverses the trajectory tree in depth-first order, and propagates the necessary derivatives backward through the tree recursively. Next, we derive the remaining core function of Algorithm 2, which performs the second-order approximation to the value function, and returns the derivatives and control updates to be propagated backward through the trajectory tree.

1) Backward control updates and derivatives: Dynamic programming over the trajectory tree requires differentiation through the value function and belief space dynamics at each observation and belief update. However, differentiation with respect to the belief state is problematic, because perturbations can push the belief off of the $|\mathcal{Z}|-1$-dimensional simplex. To solve this issue, we reparameterize the belief in terms of the unconstrained vector $\beta \in \mathbb{R}^{|\mathcal{Z}|}$, such that:

$$\mathbf{b}(z;\beta) = \frac{\exp\left(\beta\left(z\right)\right)}{\sum_{z' \in \mathcal{Z}} \exp\left(\beta\left(z'\right)\right)}. \tag{6}$$

This reparameterization allows the belief state to be represented by the vector $\mathbf{s} = [\mathbf{x}; \beta] \in \mathcal{X} \times \mathbb{R}^{|\mathcal{Z}|}$. The reparameterized belief update naturally derives from (1), such that $\beta_{t+1}(z_{t+1}) = \log(\mathbf{b}_{t+1}(z_{t+1}))$.

To derive the backward pass for PODDP, we define a function over perturbations of $\mathbf{s}$ and $\mathbf{u}$:

$$\begin{aligned}
&Q_t(\delta\mathbf{s}, \delta\mathbf{u}) = \\
&\sum_{z \in \mathcal{Z}} \mathbf{b}(z;\beta{+}\delta\beta)\big[l(\mathbf{x}{+}\delta\mathbf{x}, z, \mathbf{u}{+}\delta\mathbf{u}) \\
&\quad + V_{t+1}\big(\mathbf{x}_{ML}, h(\mathbf{o}_{ML}, \mathbf{x}_{ML}, \mathbf{x}{+}\delta\mathbf{x}, \mathbf{u}{+}\delta\mathbf{u}, \mathbf{b}(\beta{+}\delta\beta))\big)\big] \\
&- \sum_{z \in \mathcal{Z}} \mathbf{b}(z;\beta)\big[l(\mathbf{x}, z, \mathbf{u}) \\
&\qquad\qquad + V_{t+1}\big(\mathbf{x}_{ML}, h(\mathbf{o}_{ML}, \mathbf{x}_{ML}, \mathbf{x}, \mathbf{u}, \mathbf{b}(\beta))\big)\big], \quad (7)
\end{aligned}$$

where we have implicitly decomposed $\delta\mathbf{s}$ into $\delta\mathbf{x}$ and $\delta\beta$, and where $\mathbf{o}_{ML}$ and $\mathbf{x}_{ML}$ are generated according to the MLO assumption defined in lines 12-14 of Algorithm 1.

We take a second order expansion of (7):

$$\begin{aligned}
Q(\delta\mathbf{s}, \delta\mathbf{u}) &\approx \tilde{Q}(\delta\mathbf{s}, \delta\mathbf{u}) \\
&= \mathbf{1}^T Q_{\mathbf{s}}^T \delta\mathbf{s} + \mathbf{1}^T Q_{\mathbf{u}}^T \delta\mathbf{u} + \delta\mathbf{s}^T Q_{\mathbf{su}} \delta\mathbf{u} \\
&\quad + \frac{1}{2}\delta\mathbf{s}^T Q_{\mathbf{ss}} \delta\mathbf{s} + \frac{1}{2}\delta\mathbf{u}^T Q_{\mathbf{uu}} \delta\mathbf{u},
\end{aligned}$$

where each term denotes a first- or second-derivative with respect to the subscripted variables. We present the first derivatives in the main text to show their structure; the Hessians are derived in Supplementary. We note that in this work, we employ the standard iLQR approach of discarding the Hessians of the dynamics [3]. Using the following abbreviations: $\mathbf{b}_z = \mathbf{b}(z;\beta)$; $l_z = l(\mathbf{x}, \mathbf{u}, z)$; $s'_z = [\mathbf{x}_{ML}; h(\mathbf{o}_{ML}, \mathbf{x}_{ML}, \mathbf{x}, \mathbf{u}, \mathbf{b}(\beta))]$, and $V'_z = V\big(\mathbf{x}_{ML}, h(\mathbf{o}_{ML}, \mathbf{x}_{ML}, \mathbf{x}, \mathbf{u}, \mathbf{b}(\beta))\big)$, we have:

$$\begin{aligned}
Q_{\mathbf{s}} &= \sum_{z \in \mathcal{Z}} \left[\frac{\partial \mathbf{b}_z}{\partial \delta\mathbf{s}}\left(l_z + V'_z\right) + \mathbf{b}_z\left(\frac{\partial l_z}{\partial \delta\mathbf{s}} + \frac{\partial s'_z}{\partial \delta\mathbf{s}}^{\mathsf{T}} \frac{\partial V'_z}{\partial s'_z}\right)\right] \\
Q_{\mathbf{u}} &= \sum_{z \in \mathcal{Z}} \left[\mathbf{b}_z\left(\frac{\partial l_z}{\partial \delta\mathbf{u}} + \frac{\partial s'_z}{\partial \delta\mathbf{u}}^{\mathsf{T}} \frac{\partial V'_z}{\partial s'_z}\right)\right].
\end{aligned}$$

Although $\partial\mathbf{b}_z/\partial\delta\mathbf{s}$ involves differentiating the belief $\mathbf{b}_z$, the reparameterization in (6) makes these derivatives well-behaved near the simplex boundary, where the derivatives take on small values for extremal beliefs, and small perturbations $\delta\mathbf{s}$ do not violate the simplex constraint. The $\partial s'_z/\partial\delta\mathbf{s}$ and $\partial s'_z/\partial\delta\mathbf{u}$ terms involve differentiating through the dynamics, observation model, and belief update. The $\partial V'_z/\partial s'_z$ terms are the backward derivatives propagated within the Δ argument in Algorithm 2; we discuss how they are computed in Supplementary.

The optimal control modification $\delta\mathbf{u}^*$ for belief state perturbation $\delta\mathbf{s}$ is computed by minimizing the quadratic model $\tilde{Q}$:

$$\delta\mathbf{u}^*(\delta\mathbf{s}) = \arg\min_{\mathbf{u}} \tilde{Q}(\delta\mathbf{s}, \delta\mathbf{u}) = k + K\delta\mathbf{s}, \tag{8}$$

where $k = -Q_{\mathbf{uu}}^{-1} Q_{\mathbf{u}}$ is an open-loop modification to be applied in the forward pass, and $K = -Q_{\mathbf{uu}}^{-1} Q_{\mathbf{us}}$ is a linear closed-loop feedback gain.

C. Hierarchical PODDP

Because each node in the trajectory tree has $|\mathcal{Z}|$ successor nodes, the tree has size $(|\mathcal{Z}|^T - 1)/(|\mathcal{Z}| - 1) = \mathcal{O}(|Z|^T)$. This exponential growth is manageable for short horizons ($T < 5$), but for longer horizons it is infeasible. However, branching on every timestep may be unnecessary for several reasons. First, many robotics systems have high control frequency, but much lower state estimation frequency, particularly for sensor fusion from multiple modalities (i.e., cameras, LIDAR, etc.) In this case, it makes sense to align the observation timesteps in the planner with those when observations are expected to occur in the system. Second, planning with a lower observation frequency than that of the actual system can yield trajectories which take observation contingencies into account, but are more conservative than those which observe at every timestep.

To derive the hierarchical PODDP algorithm, we follow the derivation above, but partition the trajectory into a set of k segments indexed by $\tau_0 = 0, \tau_1, \ldots, \tau_k = T$. We define the value of the belief state at the beginning of a segment similarly to (5), but we now accumulate the cost over $\tau_{i+1} - \tau_i$ steps, and take the expected value of the belief state at the end of

$\tau_{i+1} - \tau_i$ steps:

$$V_{\tau_i}(\mathbf{x}_{\tau_i}, \mathbf{b}_{\tau_i}) = \min_{\mathbf{u}_{\tau_i:\tau_{i+1}-1}} \Bigg[\sum_{t=\tau_i}^{\tau_{i+1}-1} l(\mathbf{x}_t, \mathbf{b}_{\tau_i}, \mathbf{u}_t) + \mathbb{E}_{\mathbf{o}_{\tau_{i+1}-1}, \mathbf{x}_{\tau_{i+1}}} \left[V_{\tau_i+1}(\mathbf{x}_{\tau_{i+1}}, \mathbf{b}_{\tau_{i+1}}) \right] \Bigg]. \quad (9)$$

The second-order expansion can be taken similarly to before, but now with respect to perturbations of each segment. Hierarchical dynamic programming can be further optimized by applying standard DDP recursions to each step of a segment. Our experiments use hierarchical PODDP with $k = 3$.

V. Results

Our experiments analyze the performance of PODDP in three environments, designed to test the ability of PODDP to plan under uncertainty about goal locations, environment dynamics, and other agents' intentions, respectively. We compare PODDP against two baselines. The first baseline, "Maximum-likelihood DDP" (MLDDP), assumes the latent state with the highest probability is the true latent state, and runs standard DDP. At each observation point, it replans based on the updated most-likely belief. The second baseline, "Probability weighted DDP" (PWDDP), minimizes the expected cost of a control sequence with respect to the current belief – this is straightforward to implement using a version of (9), with $k = 1$ and τ_k equal to the horizon length. All runtimes were evaluated using a 3.0GHz Intel Xeon processor.

A. Experiment 1: Planning under cost function uncertainty

Our first experiment tests PODDP in a scenario in which the location of a goal is unknown, and determined by the latent world state. The environment is structured as a "T-Maze": a long corridor (surrounded by high cost regions), which splits left and right at the end. A binary latent state determines whether the goal is on the *Left* or *Right*. Goal costs which increase quadratically with the distance from the true goal location induce the agent to move to the goal as quickly as possible. Figs. 3(a,b) show the environment with a contour plot of the location cost overlaid, and goal locations marked with X's. The agent is a simulated vehicle with non-holonomic bicycle dynamics (see Supplementary). The observation function generates a Gaussian random variable conditioned on the latent state z: the mean is -1 if $z =$ *Left* and 1 if $z =$ *Right*. The uncertainty of the observation decreases as the vehicle moves to the end of the maze; this uncertainty is parameterized by a smooth function which outputs the variance of the distributions, illustrated by the background gradient in Fig. 3(a) (see Supplementary for details).

Fig. 3(a) shows a trajectory tree optimized by PODDP, starting from the belief $b(z =$ *Left*$) = 0.51$. The tree contains a contingency plan for all possible maximum-likelihood outcome sequences, conditioned on the latent state values. Fig. 3(b) shows 100 executed trajectories with uncertainty level $= 9.0$, sampling observations and state transitions from their true distributions (these trajectories were used to compute the average cost for uncertainty level $= 9.0$ in Fig. 3(c)), and replanning with a reducing horizon after each observation. Among the executed trajectories are some in which the agent first moves to one side, then crosses back to seek the goal on the other side. These correspond to "bad" observations which indicate the incorrect latent state; Fig. 3(a) shows that PODDP plans for these contingencies, and Fig. 3(b) shows that it handles them gracefully, by responding conservatively to noisy observations so that recovery is possible following later, better observations. Executed trajectories of MLDDP and PWDDP are shown in Supplementary; both methods exhibit poor qualitative performance in this task.

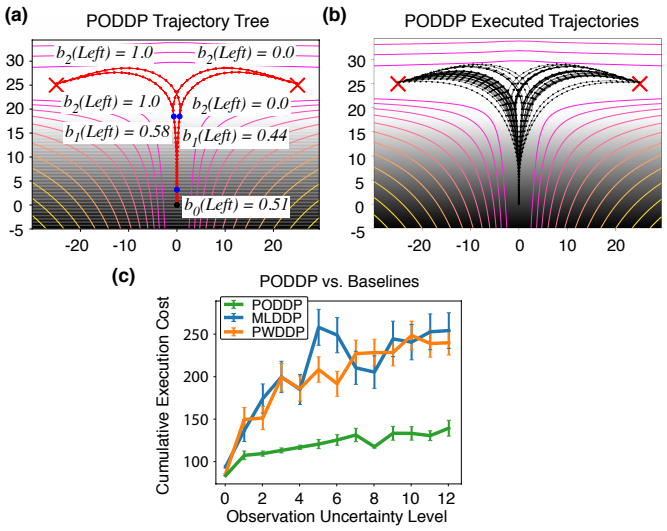

Fig. 3: Experiment 1 results. (a) Visualization of the PODDP trajectory tree in the T-Maze environment. (b) 100 sampled PODDP executions, used to compute the datapoint in (c) for observation uncertainty level $= 9.0$. See Supplementary for comparable executions of MLDDP and PWDDP. (c) Mean cumulative cost and variance of PODDP are lower than those of MLDDP and PWDDP across thirteen different observation uncertainty levels. Error bars show standard error.

Fig. 3(c) compares the average cumulative cost incurred by PODDP versus two baseline models over 100 sampled executions in each of thirteen environments, each with a different level of observation uncertainty. PODDP outperforms both baselines, and has lower variance.

Table I shows the results of a targeted analysis on the mean cumulative cost incurred by each model, and the average planning time of each model over 1000 executions for observation uncertainty level $= 9.0$. PODDP incurred significantly less mean cumulative cost than MLDDP ($t(1998) = 16.1, p < 0.000001$, and PODDP also incurred significantly less mean cumulative cost than PWDDP ($t(1998) = 17.4, p < 0.000001$). The mean cumulative costs incurred by MLDDP and PWDDP were also significantly different ($t(1998) = 2.2, p = 0.03$). The planning time for PODDP (i.e., time to optimize the initial trajectory tree in Fig. 3(a)) is substantially higher than those of the baselines, suggesting a tradeoff between computational complexity and optimality. However,

with this extra computation, PODDP is able to compute a realistic contingency plan over a long horizon, while MLDDP and PWDDP rely on frequent replanning to react to the dynamics of belief updating over the task horizon.

TABLE I: Mean cumulative cost and planning time (*seconds*) to convergence (standard error in parentheses) incurred by each model in Experiment 1 over 1000 samples.

	PODDP	**MLDDP**	**PWDDP**
cost	134.0 (2.5)	248.6 (6.6)	230.7 (4.9)
time	1.61 (0.00)	0.48 (0.00)	0.24 (0.00)

B. Experiment 2: Planning under dynamical mode uncertainty

Our second experiment tests whether PODDP can plan in the belief space over uncertain, partially observable dynamical modes of the environment. In this scenario, shown in Figs. 4(a,b), a vehicle with non-holonomic bicycle dynamics is moving toward a goal (marked by an X) over rough terrain (e.g., "mud"), which exerts a resistive force while the vehicle is moving (see Supplementary for details), imposing cost due to the additional force required to maintain a constant velocity. A binary latent state determines the smoothness of the terrain to the right of the vehicle: When the latent state $z = \textit{Smooth}$, the terrain to the right exerts low resistive force; when $z = \textit{Rough}$, the terrain to the right is rough, with high resistive force equal to that on the left. Figs. 4(a,b) show the gradient from rough to smooth terrain going from left to right when the latent state is *Smooth*.

The only source of information about the latent state comes from observing the dynamics themselves via the state sequence. This presents a challenging planning problem: exploring the environment to infer the value of z requires a costly detour right into the potentially smooth area, but the payoff is large if the agent can learn that the terrain is smooth and reduce cost thereafter.

Fig. 4(a) shows that PODDP plans an exploratory policy to learn the value of z. The planned trajectory, starting from the belief $b(z = \textit{Smooth}) = 0.49$, immediately moves to the right to gain information about z; the first observation yields strong information about z, and the beliefs become near certain, which the conditional plan can then exploit either by veering into the smooth area, or by heading directly through the mud to the goal location. Fig. 4(b) shows 100 sampled executions through the rough terrain environment, demonstrating the robustness of the planned PODDP trajectory tree. Executed trajectories of MLDDP and PWDDP are shown in Supplementary; it is qualitatively apparent that both methods fail to explore the environment, and are therefore unable to exploit the potential *Smooth* terrain as quickly as PODDP.

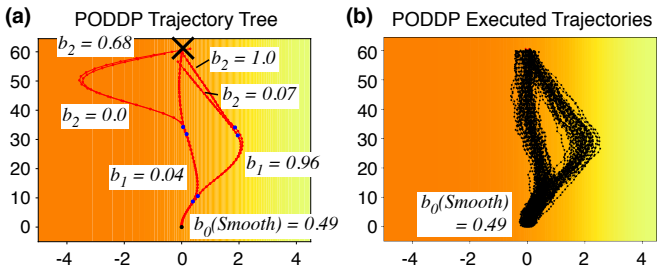

Fig. 4: Experiment 2 results. (a) Visualization of the PODDP trajectory tree in the *Rough* Terrain environment. (b) 100 sampled PODDP executions. See Supplementary for comparable executions of MLDDP and PWDDP.

Table II reports the mean cumulative cost incurred by each model, and the average planning time of each model over 1000 executions with $b_0(\textit{Smooth}) = 0.49$. PODDP incurred significantly lower mean cumulative cost than MLDDP, $(t(1998) = 4.1, p = 0.00005)$, and PODDP also incurred significantly less mean cumulative cost than PWDDP $(t(1998) = 3.9, p = 0.00009)$. The mean cumulative costs incurred by MLDDP and PWDDP were not significantly different $(t(1998) = 0.89, p = 0.37)$. The planning time for PODDP (i.e., time to optimize the trajectory tree in Fig. 4(a)) is again higher than those of the baselines, but this tradeoff may be acceptable when minimizing costs via exploration is a high priority.

TABLE II: Mean cumulative cost and planning time (*seconds*) to convergence (standard error in parentheses) incurred by each model in Experiment 2 over 1000 samples.

	PODDP	**MLDDP**	**PWDDP**
cost	220.3 (1.2)	226.4 (0.8)	228.0 (1.5)
time	0.90 (0.00)	0.27 (0.00)	0.34 (0.00)

C. Experiment 3: Latent intention-aware interactive lane changing

This experiment tested the ability of PODDP to plan trajectories through a belief state which includes the latent intentions of other agents, and dynamics which capture agents' intention-dependent actions. This scenario adds another vehicle to the state space, parameterized by a longitude and velocity (the planner vehicle again has bicycle dynamics). The other vehicle dynamics are modeled using a modified Intelligent Driver model (IDM) that considers the leading vehicle from another lane (see Supplementary for details). The latent state represents whether the other driver is *Nice* or *Aggressive*. If the other driver is *Nice*, it is assumed to have a lower desired speed, and to slow down for others. If the other driver is *Aggressive*, it is assumed to have a higher desired speed, and to not slow down for others.

Fig. 5(a) shows that PODDP can plan in the belief space over the other vehicle's latent state, and can construct a contingency plan to change lanes ahead of the other vehicle if it is inferred to be *Nice*, or change lanes behind the other vehicle if it is inferred to be *Aggressive*. Fig. 5(b) and (c) show the successful execution (black past trajectories) of these plans (red future trajectories). PWDDP also succeeds at changing lanes ahead of the *Nice* driver, and changing lanes behind the *Aggressive* driver. However, as shown in Table III, over 1000 sample executions, PWDDP incurred significantly higher cost than both PODDP $t(1998) = 12.1, p < 0.000001$ and MLDDP $t(1998) = 8.2, p < 0.000001$. In contrast, MLDDP fails to pass the *Nice* driver, and always changes lanes behind

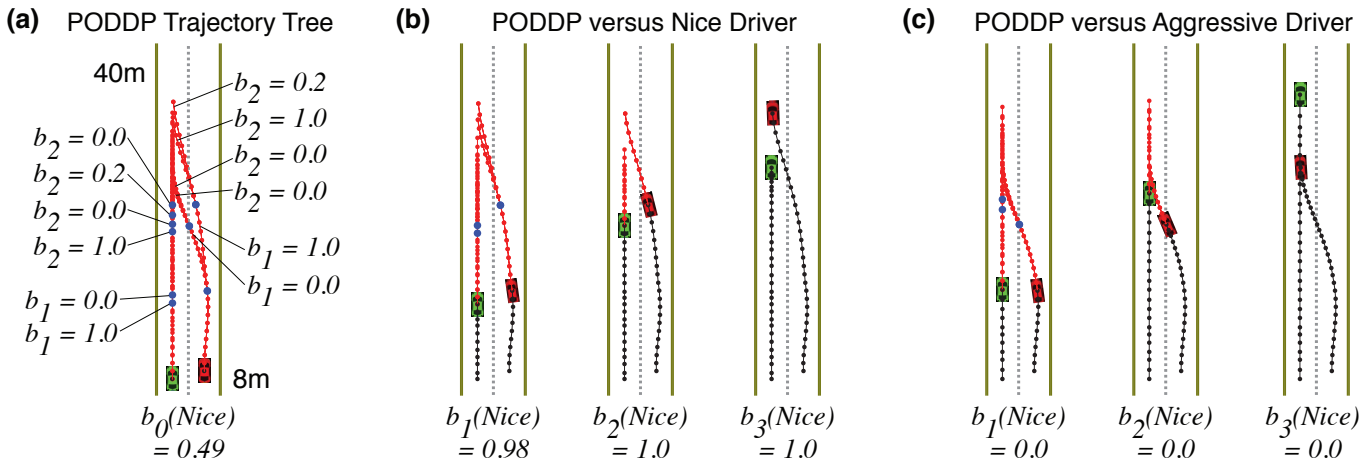

Fig. 5: Experiment 3 results. (a) Visualization of the PODDP trajectory tree in the Lane Change environment. (b) Sampled PODDP execution against a *Nice* driver. PODDP successfully merges ahead of the other vehicle. (c) Sampled PODDP execution against an *Aggressive* driver. PODDP successfully merges behind the other vehicle. See Supplementary for videos of executed trajectories for PODDP, MLDDP and PWDDP.

both *Nice* and *Aggressive* drivers, and incurs significantly higher cost than PODDP ($t(1998) = 2.4, p < 0.02$). This is because the maximum likelihood initial belief is *Aggressive*, which leads MLDDP to immediately decelerate, losing the chance to pass. To provide a fair comparison, we reran 1000 sample executions starting from $b(z = Nice) = 0.51$. With this prior, MLDDP succeeds at passing the *Nice* driver, and changing lanes behind the *Aggressive* driver, but incurs a higher mean cumulative cost, as shown in Table III. We also ran PODDP and PWDDP in this condition; as expected, the mean cumulative costs incurred were not significantly different than with the other prior.

The planning time for PODDP to compute the initial plan (i.e., the trajectory tree in Fig. 5(a)), shown in Table III, is substantially greater than those of MLDDP or PWDDP. However, Table III shows that the *replanning time* for PODDP – the total time spent replanning after each of the two observations during the task – is not qualitatively higher than the replanning time of the baselines. This suggests that although PODDP is more expensive to run initially, the latency of replanning (with warm-start) is comparable to that of the other algorithms due to the quality of the initial solution.

TABLE III: Mean cumulative cost, and planning and replanning time (*seconds*) to convergence (standard error in parentheses) incurred by each model in Experiment 3 over 1000 samples.

	$b_0(Nice) = 0.49$			$b_0(Nice) = 0.51$
	PODDP	**MLDDP**	**PWDDP**	**MLDDP**
cost	126.9 (0.8)	131.6 (1.7)	153.8 (2.0)	142.5 (1.9)
plan time	1.19 (0.002)	0.13 (0.002)	0.14 (0.002)	0.12 (0.002)
replan time	0.069 (0.000)	0.052 (0.001)	0.064 (0.001)	0.080 (0.001)

VI. Conclusion

We presented the PODDP algorithm for planning in POMDPs with continuous states, actions, and observations, nonlinear dynamics, and partially observable discrete latent states. PODDP is practical for many classes of problems: We demonstrated that it can perform belief-space planning in tasks with uncertainty about the (1) cost function, (2) dynamics, or (3) latent intentions of other agents.

There are many interesting directions for further research. In our formulation, the dynamics of the latent state of the system are assumed to be independent of the continuous state and control variables. Removing this limitation could allow application to systems with discrete variables of interest, such as contact variables in manipulation or locomotion[2]. Extending PODDP to allow partially observable continuous as well as discrete states could enable even broader applicability of continuous belief-space trajectory optimization methods.

Our formulation relied on a "maximum likelihood outcomes" assumption to approximate the belief dynamics of the system. This was sufficient for our algorithm to plan exploratory trajectories that were robust to noisy observations. Future research will explore sampling-based methods to improve performance by better approximating the belief dynamics.

We proposed a hierarchical decomposition to manage the exponential complexity of the naive algorithm in the observation frequency; aggressively limiting the observation horizon to a single future observation yields linear scaling in the number of latent states, and may be sufficient for some applications [33]. Although in our experiments the planning time of PODDP was slower than that of simple baselines, the method produces a more global policy in the belief space, which could remain valid over a longer horizon, with less need for frequent replanning, and faster replanning when it is needed. An advantage of our framework is that it provides tunable parameters for engineers to manage these tradeoffs.

[2]We thank an anonymous reviewer for this suggestion, and two other anonymous reviewers for helpful feedback and suggestions.

References

[1] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[2] Nathan Ratliff, Matthew Zucker, J. Andrew (Drew) Bagnell, and Siddhartha Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2009.

[3] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[4] Jur van den Berg, Sachin Patil, and Ron Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *International Journal of Robotics Research*, 31(11):1263–1278, 2012.

[5] Sachin Patil, Gregory Kahn, Michael Laskey, John Schulman, Ken Goldberg, and Pieter Abbeel. Scaling up Gaussian belief space planning through covariance-free trajectory optimization and automatic differentiation. In *Algorithmic Foundations of Robotics XI*, pages 515–533, 2015.

[6] Hanna Kurniawati, David Hsu, and Wee Sun Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems (RSS)*, 2008.

[7] Nan Ye, Adhiraj Somani, David Hsu, and Wee Sun Lee. DESPOT: Online POMDP planning with regularization. *Journal of Artificial Intelligence Research*, 58:231–266, 2017.

[8] Dimitri P. Bersekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, Massachusetts, 4 edition, 2017.

[9] Karl Johan Astrom. *Introduction to Stochastic Control Theory*. Academic Press, 1970.

[10] David H. Jacobson and David Q. Mayne. *Differential dynamic programming*. Elsevier, New York, 1970.

[11] Emanuel Todorov and Weiwei Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference (ACC)*, pages 300–306, 2005.

[12] Robert Platt Jr., Russ Tedrake, Leslie Kaelbling, and Tomas Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Robotics: Science and Systems (RSS)*, 2010.

[13] Tom Erez and William D. Smart. A scalable method for solving high-dimensional continuous POMDPs using local approximation. In *Twenty-Sixth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 160–167, 2010.

[14] Jur van den Berg, Pieter Abbeel, and Ken Goldberg. LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *International Journal of Robotics Research*, 30(7):895–913, 2011.

[15] Adam Bry and Nicholas Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[16] Ali-akbar Agha-mohammadi, Suman Chakravorty, and Nancy M. Amato. FIRM: Feedback controller-based information-state roadmap - A framework for motion planning under uncertainty. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4284–4291, 2011.

[17] Lydia E. Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[18] Rowan McAllister and Carl Edward Rasmussen. Data-efficient reinforcement learning in continuous state-action Gaussian-POMDPs. In *Advances in Neural Information Processing Systems 30*, pages 2040–2049, 2017.

[19] Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for POMDPs. In *35th International Conference on Machine Learning*, pages 2117–2126, 2018.

[20] Ajinkya Jain and Scott Niekum. Belief space planning under approximate hybrid dynamics. In *Robotics: Science and Systems (RSS) Workshop on POMDPs in Robotics*, 2017.

[21] Haruki Nishimura and Mac Schwager. SACBP: Belief space planning for continuous-time dynamical systems via stochastic sequential action control. In *Algorithmic Foundations of Robotics XIII*, 2019.

[22] Emma Brunskill, Leslie Kaelbling, Tomas Lozano-Perez, and Nicholas Roy. Continuous-state POMDPs with hybrid dynamics. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2008.

[23] Josep M. Porta, Nikos Vlassis, Matthijs T.J. Spaan, and Pascal Poupart. Point-based value iteration for continuous POMDPs. *Journal of Machine Learning Research*, 7:2329–2367, 2006.

[24] David Silver and Joel Veness. Monte-carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems 23*, pages 2164–2172, 2010.

[25] Panpan Cai, Yuanfu Luo, David Hsu, and Wee Sun. Lee. HyP-DESPOT: A hybrid parallel algorithm for online planning under uncertainty. In *Robotics: Science and Systems (RSS)*, 2018.

[26] Zachary N. Sunberg, Christopher Ho, and Mykel J. Kochenderfer. The value of inferring the internal state of traffic participants for autonomous freeway driving. In *Control Conference (ACC)*, 2017.

[27] Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. Continuous upper confidence trees. In *Learning and Intelligent Optimization*, pages 433–445, 2011.

[28] Konstantin M. Seiler, Hanna Kurniawati, and Surya P. N. Singh. An online and approximate solver for POMDPs with continuous action space. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2290–2297, 2015.

[29] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[30] Vadim Indelman, Luca Carlone, and Frank Dellaert. Planning in the continuous domain: A generalized belief space approach for autonomous navigation in unknown environments. *International Journal of Robotics Research*, 34(7):849–882, 2015.

[31] Haoyu Bai, Shaojun Cai, Nan Ye, David Hsu, and Wee Sun Lee. Intention-aware online POMDP planning for autonomous driving in a crowd. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 454–460, 2015.

[32] Dorsa Sadigh, S. Shankar Sastry, Sanjit A. Seshia, and Anca D. Dragan. Planning for autonomous cars that leverage effects on human actions. In *Robotics: Science and Systems (RSS)*, 2016.

[33] Jason Hardy and Mark Campbell. Contingency planning over probabilistic obstacle predictions for autonomous road vehicles. *IEEE Transactions on Robotics*, 29(4):913–929, 2013.

[34] Enric Galceran, Alexander G. Cunningham, Ryan M. Eustice, and Edwin Olson. Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction: Theory and experiment. *Autonomous Robots*, 41(6):1367–1382, August 2017.

[35] Jaime F. Fisac, Eli Bronstein, Elis Stefansson, Dorsa Sadigh, S. Shankar Sastry, and Anca D. Dragan. Hierarchical game-theoretic planning for autonomous vehicles. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[36] Peter Trautman and Andreas Krause. Unfreezing the robot: Navigation in dense, interacting crowds. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 797–803, 2010.

[37] Wilko Schwarting, Alyssa Pierson, Javier Alonso-Mora, Sertac Karaman, and Daniela Rus. Social behavior for autonomous vehicles. *Proceedings of the National Academy of Sciences*, 2019.

[38] Dorsa Sadigh, S. Shankar Sastry, Sanjit A. Seshia, and Anca Dragan. Information gathering actions over human internal state. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 66–73, 2016.

[39] Sylvie C.W. Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. POMDPs for robotic tasks with mixed observability. In *Robotics: Science and Systems (RSS)*, 2009.

[40] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

APPENDIX A
DERIVATION OF HESSIANS FROM THE BACKWARD PASS

PODDP computes a second-order approximation $\tilde{Q}$ to the Q-function defined in the main text, by taking first- and second-derivatives of the dynamics and cost functions with respect to perturbations of the belief state δs and controls δu. For reasons of space and brevity, we derive the Hessian matrices here.

Note that following the standard iLQR approach [3], we discard the terms involving Hessians of the dynamics $\frac{\partial^2 s'_z}{\partial \delta s^2}$, $\frac{\partial^2 s'_z}{\partial \delta s \partial \delta u}$, and $\frac{\partial^2 s'_z}{\partial \delta u^2}$.

The Hessians of the $\tilde{Q}$-function are:

$$Q_{ss} = \sum_{z \in \mathcal{Z}} \left[\frac{\partial^2 b_z}{\partial \delta s^2} \left(l_z + V'_z\right) + 2 \frac{\partial b_z}{\partial \delta s} \left(\frac{\partial l_z}{\partial \delta s} + \frac{\partial s'_z}{\partial \delta s}^\mathsf{T} \frac{\partial V'_z}{\partial s'_z} \right)^\mathsf{T} + b_z \left(\frac{\partial^2 l_z}{\partial \delta s^2} + \frac{\partial s'_z}{\partial \delta s}^\mathsf{T} \frac{\partial^2 V'_z}{\partial s'^2_z} \frac{\partial s'_z}{\partial \delta s} + \frac{\partial V'_z}{\partial s'_z}^\mathsf{T} \frac{\partial^2 s'_z}{\partial \delta s^2} \right) \right]$$

$$Q_{su} = \sum_{z \in \mathcal{Z}} \left[\frac{\partial b_z}{\partial \delta s} \left(\frac{\partial l_z}{\partial \delta u} + \frac{\partial s'_z}{\partial \delta u}^\mathsf{T} \frac{\partial V'_z}{\partial s'_z} \right)^\mathsf{T} + b_z \left(\frac{\partial^2 l_z}{\partial \delta s \partial \delta u} + \frac{\partial s'_z}{\partial \delta s}^\mathsf{T} \frac{\partial^2 V'_z}{\partial s'^2_z} \frac{\partial s'_z}{\partial \delta u} + \frac{\partial V'_z}{\partial s'_z}^\mathsf{T} \frac{\partial^2 s'_z}{\partial \delta s \partial \delta u} \right) \right]$$

$$Q_{uu} = \sum_{z \in \mathcal{Z}} \left[b_z \left(\frac{\partial^2 l_z}{\partial \delta u^2} + \frac{\partial s'_z}{\partial \delta u}^\mathsf{T} \frac{\partial^2 V'_z}{\partial s'^2_z} \frac{\partial s'_z}{\partial \delta u} + \frac{\partial V'_z}{\partial s'_z}^\mathsf{T} \frac{\partial^2 s'_z}{\partial \delta u^2} \right) \right]$$

APPENDIX B
DERIVATION OF VALUE FUNCTION RECURSION FROM THE BACKWARD PASS

We can plug δu^* from (8) in the main text back into $\tilde{Q}$, to calculate the approximate quadratic model of the value function V:

$$\begin{aligned} \Delta V &\approx -\frac{1}{2} k^\mathsf{T} Q_{uu} k \\ \frac{\partial V}{\partial s} &\approx Q_s - \frac{1}{2} K^\mathsf{T} Q_{uu} k \\ \frac{\partial^2 V}{\partial s^2} &\approx Q_{ss} - \frac{1}{2} K^\mathsf{T} Q_{uu} K. \end{aligned} \tag{10}$$

In Algorithm 2, the OPTIMIZECONTROL function returns $\Delta = \left\langle V, \frac{\partial V}{\partial s}, \frac{\partial^2 V}{\partial \delta s^2} \right\rangle$.

APPENDIX C
SUPPLEMENTARY EXPERIMENTAL METHODS AND RESULTS

A. Dynamic Bicycle Model

Agents in all experiments follow a 4-dimensional dynamic bicycle model. The state is defined as $[x, y, \phi, v]$, where x is the longitudinal position, y is the longitudinal position, ϕ is the orientation, and v is the velocity. The control is defined as $[\omega, a]$, where ω is the steering angle, and a is the longitudinal acceleration.

The dynamics are:

$$\begin{aligned} x(t+\delta t) &= x(t) + v\cos(\phi)\delta t + \epsilon_x \\ y(t+\delta t) &= y(t) + v\sin(\phi)\delta t + \epsilon_y \\ \phi(t+\delta t) &= \phi(t) + \frac{v}{L}\tan(\omega)\delta t + \epsilon_\phi \\ v(t+\delta t) &= v(t) + a\delta t + \epsilon_v, \end{aligned}$$

where L is the vehicle length, and ϵ_x, ϵ_y, ϵ_ϕ, and ϵ_v are additive Gaussian noise terms.

B. Experiment 1

1) Scenario formulation: In the T-Maze environment, the observability of the latent state improves as the agent moves longitudinally down the corridor. Given an uncertainty level ξ the standard derivation of the observation is formulated as

$$\sigma(y) = 0.1 + (s(-(y-c))/c) \cdot \xi$$

where y is the coordinate of the agent along the hallway, $c = 18.0$ defines the border of the region with noisy observations, and $s(x) = \left(\sqrt{x^2+1} + x\right)/2$. An observation o conditioned on the latent state z is then sampled from the observation function given by:

$$o \sim O(\cdot|z, y) = \begin{cases} \mathcal{N}(-1, \sigma(y)) \text{ if } z = \textit{Left} \\ \mathcal{N}(1, \sigma(y)) \text{ if } z = \textit{Right} \end{cases}$$

where $\mathcal{N}$ is a normal distribution function. The goal on the $Left$ is located at $(-25.0, 25.0)$ and the goal on the $Right$ is located at $(25.0, 25.0)$.

2) Model evaluation: To quantitatively evaluate the model, we ran 100 executions in thirteen different environments, each with a different level of observation uncertainty in figure 3(a). The observation uncertainty levels used were $[0.0, 1.0, 2.0, \ldots, 12.0]$. Each execution sampled the ground truth world state from the prior (set to $b_0(\textit{Left}) = 0.51$), and sampled observations from the observation distribution at each observation timestep. For this experiment, the timestep was $\delta t = 0.1$, and the planning horizon was $T = 60$. We set the number of segments for hierarchical PODDP to be $k = 3$; therefore there were two observations over the planning horizon, at $\tau_1 = 20$ and $\tau_2 = 40$. All algorithms replanned after each observation. The vehicle dynamics were assumed to be deterministic, because they were independent of the latent state value, and not relevant for the task.

3) Supplementary results: Fig. 6(a) and (b) show sampled trajectories from MLDDP and PWDDP, respectively. Both algorithms fail to explore as aggressively as PODDP, which accelerates more rapidly in the beginning of the trajectory to get as reliable an observation as possible. The possible MLDDP trajectories split at each observation point, depending on the maximum-likelihood belief state following the observation. These trajectories commit strongly to the maximum-likelihood goal location, and when a bad observation occurs, they are unable to recover. The PWDDP trajectories also commit early, based on minimizing the goal costs to both locations simultaneously, weighted by their probability. Only when the initial observation is very strong can PWDDP can commit strongly.

Algorithm 1: FORWARDPASS $(x_0, b_0, U_{\text{nom}}, S_{\text{nom}}, k, K, \alpha, \mathcal{Z}, T)$

$U \leftarrow [\,]$; // initialize control map indexed by histories
$S(\text{'Root'}) \leftarrow [\mathbf{x}_0, \mathbf{b}_0]$; // initialize belief state map indexed by histories
FORWARDTREE $(\text{'Root'}, U, S, U_{\text{nom}}, S_{\text{nom}}, \varnothing, \varnothing, \alpha, \mathcal{Z}, T, 1)$; // trajectory tree recursion
return U, S ; // return updated trajectory tree
Procedure FORWARDTREE$(H, U, S, U_{nom}, S_{nom}, k, K, \alpha, \mathcal{Z}, T, d)$
  if $k(H), K(H) \neq \varnothing$ **then** // apply control updates
    $U(H) \leftarrow U_{\text{nom}}(H) + \alpha k(H) + K(H)(S(H) - S_{\text{nom}}(H))$;
  else
    $U(H) \leftarrow U_{\text{nom}}(H)$;
  for $z \in \mathcal{Z}$ **do**
    $[\mathbf{x}_H, \mathbf{b}_H] \leftarrow S(H)$;
    $\mathbf{x}_{ML} = \arg\max_{\mathbf{x}} p(\mathbf{x}|\mathbf{x}_H, U(H), z)$; // assume ML state transition
    $\mathbf{o}_{ML} = \arg\max_{\mathbf{o}} p(\mathbf{o}|\mathbf{x}_{ML}, z)$; // assume ML observation
    $\mathbf{b}' = \text{BELIEFUPDATE}(\mathbf{o}_{ML}, \mathbf{x}_{ML}, U(H), \mathbf{x}_H, \mathbf{b}_H)$;
    $S([H, z]) \leftarrow [\mathbf{x}_{ML}, \mathbf{b}']$; // append new belief state to history
    if $d < T$ **then**
      FORWARDTREE $([H, z], U, S, U_{\text{nom}}, X_{\text{nom}}, k, K, \alpha, \mathcal{Z}, T, d+1)$; // recurse

Algorithm 2: BACKWARDPASS $(U, S, \mathcal{Z}, T)$

$k, K \leftarrow [\,]$; // initialize control update maps indexed by histories
BACKWARDTREE $(\text{'Root'}, k, K, U, S, \mathcal{Z}, T, 1)$; // compute control updates recursively
return k, K ; // return control updates
Procedure BACKWARDTREE$(H, k, K, u, S, \mathcal{Z}, T, d)$
  $\Delta \leftarrow [\,]$; // initialize backward derivatives map
  for $z \in \mathcal{Z}$ **do**
    if $d < T$ **then** // recursively compute backward derivatives and updates
      $\Delta(z) \leftarrow \text{BACKWARDTREE}([H, z], k, K, U, S, \mathcal{Z}, T, d)$;
    else // set backward derivatives as empty
      $\Delta(z) \leftarrow \varnothing$;
  $k_H, K_h, \Delta_H \leftarrow \text{OPTIMIZECONTROL}(U(H), S(H), \Delta, \mathcal{Z})$;
  $k(H), K(H) \leftarrow k_H, K_H$; // update control update maps
  return Δ_H ; // return backward derivatives

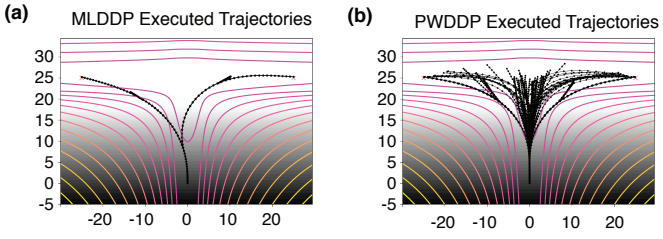

Fig. 6: Experiment 1 results. (a) 100 sampled MLDDP executions. (b) 100 sampled PWDDP executions. Compare to Main Text, Fig. 3(c).

C. Experiment 2

1) Scenario formulation: To capture the rough terrain in this experiment, we assume that the terrain exerts a location dependent resistive force on the vehicle. We model this force as a velocity-dependent deceleration r, such that:

$$r = \rho * \tanh(v),$$

which the vehicle must overcome with its own acceleration, incurring cost. The parameter ρ varies according to the location. In our scenario, ρ is high for rough terrain, and zero for smooth terrain. The transition in our environment is sigmoidal in the x-dimension.

For this experiment, the timestep was $\delta t = 0.1$, and the planning horizon was $T = 60$. We set the number of segments for hierarchical PODDP to be $k = 3$; therefore there were two observations over the planning horizon, at $\tau_1 = 20$ and $\tau_2 = 40$.

In this scenario, because all information about the latent state comes from the state transitions, we assume additive

Gaussian noise at each timestep for all state variables to make the belief state dynamics nontrivial.

2) Supplementary results: Fig. 7(a) and (b) show sampled trajectories from MLDDP and PWDDP, respectively. Both algorithms fail to explore as aggressively as PODDP, which veers left into the potentially smooth area to obtain as reliable an observation as possible. MLDDP initially assumes that $z =$ *Rough*, and heads directly for the goal location. If the maximum-likelihood belief after the first observation is $z =$ *Smooth*, it moves into the smooth region. PWDDP actually moves away from the smooth region, because this trajectory allows it to minimize distance from the goal with the same control sequence under both dynamics. This is an interesting and subtle feature of PWDDP. It also moves into the smooth region if the belief updates in favor of $z =$ *Smooth*, but it does not explore or plan for future observation contingencies.

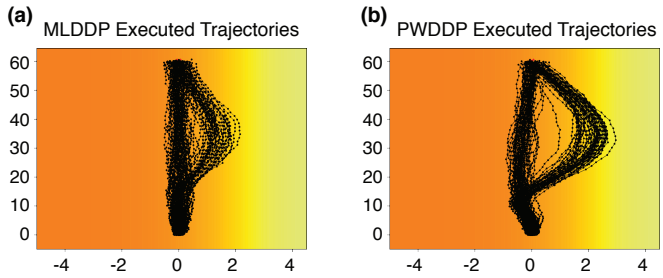

Fig. 7: Experiment 2 results. (a) 100 sampled MLDDP executions. (b) 100 sampled PWDDP executions. Compare to Main Text, Fig. 4(b).

D. Experiment 3

1) Scenario formulation: In this scenario, we assume that both vehicles start from a velocity of 10 m/s. The dynamics of the other vehicle follows a modified IDM, which shares the same acceleration control principle of the standard IDM, while considering the leading vehicle on an adjacent lane by modifying the distance metric between the agent and the leading vehicle from $s(l, a) = |y_l - y_a|$ to

$$s'(l, a) = \sigma_{c,\delta}\left(|x_l - x_a|\right) \cdot s(l, a)$$

where x is the coordinate perpendicular the the lanes and y is the coordinate along the lanes with l indicating the leading vehicle and a indicating the IDM agent, and sigmoid function $\sigma_{c,\delta}(x) = \left(1 + \exp^{-\delta(x-c)}\right)^{-1}$. For the sigmoid function σ, parameter c is assigned with the value of half the lane width, and δ is an adjustable steepness parameter.

For this experiment, the timestep was $\delta t = 0.1$, and the planning horizon was $T = 30$. We set the number of segments for hierarchical PODDP to be $k = 3$; therefore there were two observations over the planning horizon, at $\tau_1 = 10$ and $\tau_2 = 20$.

In this scenario, because all information about the latent state comes from the other vehicle's state transitions, we assume additive Gaussian noise at each timestep for all state variables to make the belief state dynamics nontrivial.

2) Supplementary results: Please see our GitHub page (https://davidqiu1993.github.io/poddp-paper) for supplementary videos.