Approximate Inference-Based Motion Planning by Learning and Exploiting Low-Dimensional Latent Variable Models

Jung-Su Ha 🆔, Hyeok-Joo Chae 🆔, and Han-Lim Choi 🆔

Abstract—This work presents an efficient framework to generate a motion plan of a robot with high degrees of freedom (e.g., a humanoid robot). High dimensionality of the robot configuration space often leads to difficulties in utilizing the widely used motion planning algorithms, since the volume of the decision space increases exponentially with the number of dimensions. To handle complications arising from the large decision space, and to solve a corresponding motion planning problem efficiently, two key concepts are adopted in this work: First, the Gaussian process latent variable model (GP-LVM) is utilized for low-dimensional representation of the original configuration space. Second, an approximate inference algorithm is used, exploiting through the duality between control and estimation, to explore the decision space and to compute a high-quality motion trajectory of the robot. Utilizing the GP-LVM and the duality between control and estimation, we construct a fully probabilistic generative model with which a high-dimensional motion planning problem is transformed into a tractable inference problem. Finally, we compute the motion trajectory via an approximate inference algorithm based on a variant of the particle filter.

Index Terms—Motion and path planning, learning from demonstration.

I. INTRODUCTION

FOR robotic motion planning, a trajectory is designed for robot states through a complex configuration space from an initial state to perform a given task. The planning problem is formulated as an optimal control (OC) problem considering the robot dynamics for a feasible motion trajectory and the cost function for the task. The optimal trajectory is reconstructed from the optimal cost-to-go function (also called the value function), which is obtained by solving the Bellman equation through dynamic programming procedures. Though these approaches guarantee the global optimality of the solution, they

are not scalable with the dimensionality of the decision space because of the curse of dimensionality: the size of the decision space increases exponentially with the number of dimensions. Sampling-based algorithms such as RRT* [1] or FMT* [2], and their variants, are also widely used in the motion planning literature. These algorithms simultaneously construct and extend the approximate state space (represented in a graph or a tree structure), and then update the approximate solution. They are applicable to medium-sized problems but it is almost impossible to extend the graph into extremely high-dimensional space and obtain the solution without limiting the sampling space. Another option is a trajectory optimization based on iterative local optimization [3], [4]. These approaches approximate the problem around the current solution using the first or second order Taylor expansion and update the solution iteratively. The solution from these approaches only guarantees local optimality from its nature. For high-dimensional problems, it is often required to give a valuable initial guess for the optimization, which is then painstakingly handcrafted by a user/designer, because the problem may have many poor local optima. In addition, the local approximation for all dimensions can cause the solution procedure to diverge, which enforces heuristic techniques to be used.

Given this limitations, one research question that may arise is whether or not the decision space we need to search should be this large. It may be that the dimension of the decision space really needed to be considered is small, if there is some prior structures in decisions [5]. Think about a motion planning process of our own body; when we have decided on which type of motion we are going to make, the sequence of our poses and the configurations we really need to consider may be limited. For example, we would not consider walking on our hands when the mission is to get to a certain goal position from a start position. That is, it is reasonable to assume that, although a robot has very high degrees of freedom, the valuable configurations take up a very small portion in the original space, and that they form some sort of (low-dimensional) manifolds. This is the spirit behind the latent variable model. There have been many studies aimed at finding the embedding of manifolds in high-dimensional space within a low-dimensional latent space. Specifically, the Gaussian process latent variable model (GP-LVM) assumes that there exists a probabilistic model for mapping from low-dimensional latent space to high-dimensional observation space, and then finds the mapping and the corresponding latent space [6]. Moreover, the Gaussian process dynamical model (GPDM) extends

the GP-LVM such that dynamics in the observation space can be represented by that of the latent space [7]. Human motions in 50+dimensional configuration space have successfully been embedded into 2-3 dimensional latent space by GPDM, and the model has been utilized to generate new motions of a human character [7]–[9] and for 3-D tracking of people [10]. Recently, with the successes in deep learning, there have been many attempts utilizing deep neural network to construct the latent variable model; one of the most popular algorithms is the variational autoencoder (VAE) [11]. The algorithm has also been applied to robotics applications: [12] exploited the idea of VAEs to embed dynamic movement primitives into the latent space. [13] used the conditional VAEs to learn a non-uniform sampling methodology of a sampling-based motion planning algorithm. [14] constructed a latent dynamical system from videos and utilized the learned model to compute an optimal control policy in a model-predictive control scheme. In [15], [16], we presented a variational inference method to train latent models parameterized by neural networks and utilized the learned model for planning in a similar way with this work.

One theoretical concept this work extensively takes advantage is the duality between the optimal control and estimation [17]–[22]. The idea is that, if we consider an artificial binary observation whose emission probability is given by the exponential of a negative cost, an OC problem can be reformulated as an equivalent inference problem. In this case, the objective is to find the trajectory or control policy that maximizes the likelihood of the observations along the trajectory. To address the transformed inference problem, several approximate inference techniques were utilized. If transitions between time steps are made approximately Gaussian [4] with the current control policy, the resulting algorithm becomes (locally) equivalent to the iterative linear quadratic Gaussian method [3]. In path integral control approaches, particles propagated by the current approximate control policy [19], or the control policy induced by a higher-level path planner [23], are used to approximate the resulting distribution of the inference. The user-designed probability model can also be utilized as a priori to solve the transformed inference problem [21]. Especially in [22], the planning problem is converted into the inference problem for a factored graph of a continuous motion trajectory represented by Gaussian process, and an incremental inference method, Bayes Trees, is utilized to efficiently perform replanning procedures. Finding a valuable proposal distribution is essential to the efficient inference method; just like finding a valuable search-space is essential for the planning problem.

A. Overview and Contribution

This work addresses a motion planning problem of a robot with high-dimensional configuration space. The objective of the planning problem is to generate a sequence of configurations that achieves a given task and satisfies certain constraints, while being smooth in terms of the robot dynamics. Rather than solving the planning problem in the original configuration space, we construct a latent variable model with dynamics of much lower number of dimensions and then solved the problem utilizing them. The latent model with dynamics can be learned from experts' demonstration data or from robot's own experiences; this
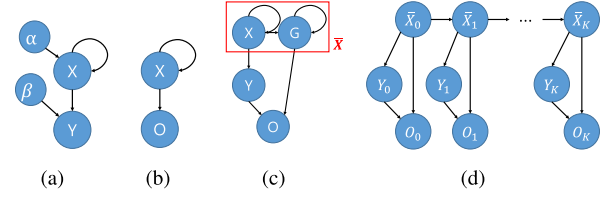

Fig. 1. (a) The GPDM compresses high-dimensional sequential data, $\mathbf{Y}$, into a low-dimensional dynamic system with $\mathbf{X}$, where α and β are hyperparameters of the model. (b) The probabilistic model for optimal control; the optimal motion plan can be computed by inferring a posterior trajectory, $\mathbf{X}$, given artificial observations, $\mathbf{O}$, encoding the cost function of control problem. (c), (d) The fully-probabilistic model for optimal motion planning with the GPDM (hyperparameters are omitted for visualization sake).

work particularly considers the cases where the demonstration data is available. The learned latent model shown in Fig. 1(a) represents a probabilistic mapping from the latent space $\mathcal{X}$ to the configuration space $\mathcal{Y}$, and of the stochastic dynamics in the latent space. A detailed description of latent variable models is given in Section II.

The model is combined with the probabilistic model for optimal control (shown in Fig. 1(b)) that is built using the duality between optimal control and Bayesian estimation. The intuition behind the duality is that the likelihood of a trajectory for an optimally controlled system is equivalent to the posterior probability when the cost related artificial observation is observed. (More details will be addressed in Section III-A.) The combined fully-probabilistic model is shown in Fig. 1(c) and 1(d). With this, we can convert the original motion planning problem into an inference problem, where the objective is to find the maximum a posteriori (MAP) trajectory given the artificial observation, $O_{1:K}$. This allows for efficient solution, because every piece of dynamic information is encoded in the low-dimensional latent space. Descriptions of the combined probabilistic model and proposed solution method are given in Section III-B and III-C, respectively. Finally, a multiscale acceleration method that increases the algorithm efficiency based on the path integral control is introduced in Section III-D. The proposed framework has several advantages; (a) it does not require knowledge of system dynamics, because this is learned from demonstration data; (b) rather than exploring the original configuration space [21], our framework uses the stochasticity of the latent space for exploration and this provides valuable search space of the motion planning problem; (c) when computing the most likely trajectory, our framework utilizes a Markov property of the resulting probability model. This provides a more efficient solution method than provided by naive optimization approaches, where the trajectory itself along all the time horizon is considered an (huge) optimization variable [7], [8].

II. CONSTRUCTING A LATENT VARIABLE MODEL

A. Gaussian Process Latent Variable Model With Dynamics

The GP-LVM [6] is a generative model that represents a probabilistic mapping from a latent space $\mathcal{X}$ to the observation space $\mathcal{Y}$. Let $\mathbf{Y} \equiv [\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_N]^T \in \mathbb{R}^{N \times D}$ be the observation matrix, where each row of the matrix represents a single high-dimensional observation of training sequential data, and $\mathbf{X} \equiv [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]^T \in \mathbb{R}^{N \times d}$ be the matrix whose rows rep-

resent corresponding latent coordinations of the observations. In the GP-LVM, the mapping is assumed to be Gaussian process with a covariance function $k_Y(\cdot,\cdot):\mathbb{R}^d\times\mathbb{R}^d\to\mathbb{R}$, and then the likelihood of the observation data is given by: $p(\mathbf{Y}|\mathbf{X},\beta)=$

$$\frac{1}{\sqrt{(2\pi)^{ND}|\mathbf{K}_Y|^D}}\exp\left(-\frac{1}{2}\mathrm{tr}\left(\mathbf{K}_Y^{-1}\mathbf{Y}\mathbf{Y}^T\right)\right),\qquad(1)$$

where $\mathbf{K}_Y\in\mathbb{R}^{N\times N}$ is a kernel matrix whose components are computed as $(\mathbf{K}_Y)_{ij}=k_Y(\mathbf{x}_i,\mathbf{x}_j)$ with kernel hyperparameters β.

If the observations are assumed to be obtained from a dynamic system, it would also be possible to construct a probabilistic dynamic model in the latent space [7]. With this model, the likelihood of the latent variable sequence is given by: $p(\mathbf{X}|\alpha)=$

$$\frac{p(\mathbf{x}_1)}{\sqrt{(2\pi)^{(N-1)d}|\mathbf{K}_X|^d}}\exp\left(-\frac{1}{2}\mathrm{tr}\left(\mathbf{K}_X^{-1}\mathbf{X}_{2:N}\mathbf{X}_{2:N}^T\right)\right),\quad(2)$$

where the kernel matrix $\mathbf{K}_X\in\mathbb{R}^{(N-1)\times(N-1)}$ is constructed by $\mathbf{X}_{1:N-1}$ with kernel hyperparameters α and $\mathbf{x}_1$ is assumed to have a Gaussian prior. Given the prior of hyperparameters, $p(\alpha)$ and $p(\beta)$, a latent variable model with stochastic dynamics is constructed by maximizing the posterior which is proportional to the following joint probability:

$$p(\mathbf{X},\mathbf{Y},\alpha,\beta)=p(\mathbf{Y}|\mathbf{X},\beta)p(\mathbf{X}|\alpha)p(\alpha)p(\beta).\qquad(3)$$

Note that the GP-LVM is a generative model that ensures smooth mapping from the latent space to the observation space, i.e., the mapping from $\mathcal{X}$ to $\mathcal{Y}$ is continuous and differentiable. Therefore, the learning process optimizes the model such that two points that are close together in the latent space are mapped to points that are close in the observation space [24]. Equivalently, two points that are far apart in the observation space cannot have close latent coordinates, which condition is called *dissimilarity preservation*, and the reverse is only guaranteed when mapping is linear. Preserving similarity is often considered more important in dimensionality reduction, because observations sparsely lie in high-dimensional space and then similar observations may contain more valuable information between them. In order to make the GP-LVM have the property of *similarity preservation*, back-constraints were introduced [24], which enforces a smooth mapping from the observation space to the latent space. For example, the kernel based regression model could be used as smooth mapping: $x_{ij}=\sum_{m=1}^N a_{jm}k(\mathbf{y}_i,\mathbf{y}_j)$, where closeness in the observation space is measured by the kernel function $k(\cdot,\cdot)$. Any differentiable mapping (e.g., neural network) could be utilized here; then, the optimization process with respect to the latent coordination $\mathbf{X}$ will simply turn into that with respect to the mapping parameters $\{a_{jm}\}$ using the chain rule. It is also possible to inject prior knowledge into latent space by restricting a structure of back-constraints [8].

B. Latent Space Dynamical System

With the constructed latent variable model, we have the following stochastic dynamics in the latent space:

$$\mathbf{x}_{k+1}=\mu_X(\mathbf{x}_k)+\Sigma_X^{1/2}(\mathbf{x}_k)\mathbf{w}_k,\qquad(4)$$

where the mean μ_X and the variance Σ_X are given by the posterior of the Gaussian process:

$$\mu_X(\mathbf{x})=\mathbf{X}_{2:N}\mathbf{K}_X^{-1}\mathbf{k}_X(\mathbf{x}),$$
$$\Sigma_X(\mathbf{x})=k_X(\mathbf{x},\mathbf{x})-\mathbf{k}_X(\mathbf{x})^T\mathbf{K}_X^{-1}\mathbf{k}_X(\mathbf{x}),\qquad(5)$$

and $\mathbf{w}_k$ is a d-dimensional standard Gaussian random noise. Here, $\mathbf{k}_X(\mathbf{x})\in\mathbb{R}^{N-1}$ is a vector of which ith element represents $k_X(\mathbf{x},\mathbf{x}_i)$. Moreover, the corresponding pose is also normally distributed:

$$\mathbf{y}_k\sim\mathcal{N}\left(\mu_Y(\mathbf{x}_k),\Sigma_Y(\mathbf{x}_k)\right),\qquad(6)$$

where the mean μ_Y and the variance Σ_Y are given by

$$\mu_Y(\mathbf{x})=\mathbf{Y}\mathbf{K}_Y^{-1}\mathbf{k}_Y(\mathbf{x}),$$
$$\Sigma_Y(\mathbf{x})=k_Y(\mathbf{x},\mathbf{x})-\mathbf{k}_Y(\mathbf{x})^T\mathbf{K}_Y^{-1}\mathbf{k}_Y(\mathbf{x}),\qquad(7)$$

and $\mathbf{k}_Y(\mathbf{x})\in\mathbb{R}^N$ is similarly defined as above. Combining (4) and (6), the graphical representation of the learned generative model is shown in Fig. 1(a).

III. OPTIMAL MOTION PLANNING IN HIGH DIMENSION VIA APPROXIMATE INFERENCE

A. Optimal Control via Inference

Consider a passive and controlled stochastic dynamics,

$$\mathbf{x}_{k+1}^{passive}\sim p_k(\cdot|\mathbf{x}_k),\qquad(8)$$
$$\mathbf{x}_{k+1}^{controlled}\sim\pi_k(\cdot|\mathbf{x}_k),\qquad(9)$$

respectively, and the cost rate

$$l_k(\mathbf{x},\pi(\cdot|\mathbf{x}))=q_k(\mathbf{x})+D_{KL}\left(\pi(\cdot|\mathbf{x})||p(\cdot|\mathbf{x})\right),\qquad(10)$$

where $q(\mathbf{x})$ is an instantaneous state cost rate that encodes a given task and $D_{KL}(\pi||p)$ is the Kullback-Leibler (KL) divergence that penalizes a deviation of the controlled dynamics from the passive one. Then, a stochastic OC problem is formulated with the total cost:

$$J^\pi=\mathbb{E}_{x'\sim\pi(\cdot|x)}\left[q_K(\mathbf{x}_K)+\sum_{k=0}^{K-1}l_k(\mathbf{x}_k,\pi_k(\cdot|\mathbf{x}_k))\right].\quad(11)$$

The above OC problem can be solved by defining the value function,

$$v_k(\mathbf{x})=\min_\pi\mathbb{E}_{x'\sim\pi(\cdot|x)}\left[q_K(\mathbf{x}_K)+\sum_{\kappa=k}^{K-1}l_\kappa(\mathbf{x}_\kappa,\pi_\kappa(\cdot|\mathbf{x}_\kappa))\right],$$

and solving the Bellman equation on it. Especially, it is known that a solution of the above OC problem satisfies the linear Bellman equation on the exponentiated value function, called the *desirability function*, $z_k(\mathbf{x})=\exp(-v_k(\mathbf{x}))$ as:

$$z_k(\mathbf{x})=\begin{cases}\exp(-q_K(\mathbf{x})), & \text{if } k=K,\\ \exp(-q_k(\mathbf{x}))\mathcal{G}[z_{k+1}(\cdot)](\mathbf{x}), & \text{otherwise,}\end{cases}\qquad(12)$$

where the linear operator $\mathcal{G}[f(\cdot)](\mathbf{x})\equiv\int p(\mathbf{x}'|\mathbf{x})f(\mathbf{x}')d\mathbf{x}'$, denotes the average value of the function f at the next time-step [25]. The optimally controlled dynamics is also obtained

as:

$$\pi_k^*(\mathbf{x}'|\mathbf{x}) = \frac{p_k(\mathbf{x}'|\mathbf{x})z_{k+1}(\mathbf{x}')}{\mathcal{G}[z_{k+1}(\cdot)](\mathbf{x})}. \tag{13}$$

Then, the probability of the trajectory $\mathbf{x}_{1:K} \equiv \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_K\}$ when the state evolves with the optimal transition dynamics π^* from the initial state $\mathbf{x}_0$ is given as: $p^*(\mathbf{x}_{1:K}|\mathbf{x}_0) \propto$

$$\exp(-q_K(\mathbf{x}_K)) \prod_{k=0}^{K-1} \exp(-q_k(\mathbf{x}_k))p_k(\mathbf{x}_{k+1}|\mathbf{x}_k). \tag{14}$$

For the detailed derivation, we refer readers to [20], [25] and the references therein.

The OC problem above can be transformed into the Bayesian inference problem. Suppose we have an artificial binary observation o_k whose emission probability is given by exponential of a negative cost, i.e.,

$$p^o(o_k = 1|\mathbf{x}_k) = \exp(-q_k(\mathbf{x}_k)). \tag{15}$$

The corresponding graphical model is shown in Fig. 1(b); the transition of the state $\mathbf{x}$ is governed by p_k and it emerges the observation o with p^o. Then, the probability of the trajectory $\mathbf{x}_{1:K}$ given the initial state $\mathbf{x}_0$ and the observation $o_k = 1, \forall k = 1, ..., K$ is given as: $p(\mathbf{x}_{1:K}|\mathbf{x}_0, o_{1:K} = 1) \propto$

$$\prod_{k=0}^{K-1} p^o(o_{k+1} = 1|\mathbf{x}_{k+1})p_k(\mathbf{x}_{k+1}|\mathbf{x}_k)$$

$$\propto \exp(-q_K(\mathbf{x}_K)) \prod_{k=0}^{K-1} \exp(-q_k(\mathbf{x}_k))p_k(\mathbf{x}_{k+1}|\mathbf{x}_k). \tag{16}$$

From (14) and (16), we observe that the OC problem (8)–(11) is closely related to a Bayesian estimation problem that infers the state trajectory $\mathbf{x}_{1:K}$ when the observation $o_{1:K} = 1$ and the initial state $\mathbf{x}_0$ is given (see Fig. 1(b)) [18], [20]. The trajectory (or state) distribution induced by the optimal control policy is equivalent to the posterior distribution of the trajectory in the inference problem. Once the inference problem is formulated, any approximate inference method, such as expectation propagation [4], particle belief propagation [21], or importance sampling [19], [23], can be utilized to solve the OC problem efficiently.

B. Motion Planning as MAP Trajectory Estimation

The framework we propose in this work combines ideas of the latent variable model and the Bayesian interpretation of an OC problem to solve motion planning problems for high-dimensional systems. The graphical representation of a combined fully-probabilistic model for high-dimensional motion planning problems is shown in Fig. 1(d). When considering locomotions, the motion data used to learn the latent variable model do not include the global position and orientation of an agent, but only recode their variations, e.g. (angular) velocity, because the learned model needs to encode the dynamics of agent's poses; once the sequence of poses are realized, the global position and orientation can be computed by integrating their variations. Let $\mathbf{g} \in \mathbb{R}^3$ be the vector of the horizontal position, (x, y), and heading angle, θ, of the robot and $(\mathbf{y})_v \in \mathbb{R}^3$ denote components corresponding to the forward/lateral velocities and

yaw rate of the robot.[1] Then, $\mathbf{g}$ also has stochastic dynamics as:

$$\mathbf{g}_{k+1} = \mathbf{g}_k + R_\theta \left(\mu_{Y_v}(\mathbf{x}_k) + \Sigma_{Y_v}^{1/2}(\mathbf{x}_k)\epsilon_k \right), \tag{17}$$

where R_θ is a rotation matrix of the z axis and $\epsilon_k \sim \mathcal{N}(0, I_3)$ follows the standard normal distribution. Together with the dynamics in the latent space, we define the augmented latent variable as $\bar{\mathbf{x}} \equiv [\mathbf{x}^T, \mathbf{g}^T]^T$, and then the arrows between the above-most nodes in Fig. 1(d) represent the temporal structure of the motion planning problem. Moreover, the latent state at each time step induces robot pose, $\mathbf{y}$ as in (6). In the planning problem, a cost function, q_k, takes robot pose $\mathbf{y}$ as well as its global position and orientation $\mathbf{g}$ as arguments and encodes specifications of a given task, e.g., collision with obstacles or distance from a goal region, etc. Therefore, $\{\mathbf{y}, \mathbf{g}\}$ induces the emission probability of an artificial observation in the proposed probabilistic model in Fig. 1(d) as:

$$p^o(o_k = 1|\mathbf{y}_k, \mathbf{g}_k) = \exp(-q_k(\mathbf{y}_k, \mathbf{g}_k)). \tag{18}$$

In this work, we define the optimal motion plan as the most likely trajectory when the robot's (stochastic) dynamics is optimally controlled. From the close relationship between (14) and (16), the problem of finding the most likely trajectory under the optimally controlled dynamics can be converted into the Bayesian estimation problem. For the transformed estimation problem, the objective is to compute the MAP trajectory $\mathbf{y}_{1:K}^*$ with given initial latent and global coordinates $\bar{\mathbf{x}}_0$ and artificial observations $o_{1:K} = 1$:

$$\{\mathbf{y}_{1:K}^*, \bar{\mathbf{x}}_{1:K}^*\} = \underset{\mathbf{y}_{1:K}, \bar{\mathbf{x}}_{1:K}}{\operatorname{argmax}} \, p(\mathbf{y}_{1:K}, \bar{\mathbf{x}}_{1:K}|\bar{\mathbf{x}}_0, o_{1:K} = 1). \tag{19}$$

C. Dynamic Programming for Computing MAP Trajectory Using Particle Filter

The estimation problem in (19) includes two sources of difficulty: long time horizon and continuous space. The first difficulty can be addressed by exploiting the Markov property of the probability model. Then, the posterior probability of the trajectory is factorized along the time axis as:

$$\{\mathbf{y}_{1:K}^*, \bar{\mathbf{x}}_{1:K}^*\}$$

$$= \underset{\mathbf{y}_{1:K}, \bar{\mathbf{x}}_{1:K}}{\operatorname{argmax}} \prod_{k=1}^{K} p^o(o_k = 1|\mathbf{y}_k, \mathbf{g}_k)p(\mathbf{y}_k|\mathbf{x}_k)p(\bar{\mathbf{x}}_k|\bar{\mathbf{x}}_{k-1}).$$

$$\tag{20}$$

If the state space is a discrete set, the estimation problem above can be solved via a simple dynamic programming (DP) procedure called the Viterbi algorithm [26]. However, the observation space $\mathcal{Y}$, as well as the latent space $\mathcal{X}$ in our problem, is continuous; so we need a scheme for approximation of distributions over the continuous space. In this work, the approximate inference algorithm introduced in [27] is adopted to address such a difficulty. The algorithm basically discretizes the continuous state space by the particles obtained from the particle filter (PF) procedure which are expected to span valid regions of the state space due to the resampling procedure. While the PF procedure builds the approximate discrete state space, the Viterbi

---

[1]$\mathbf{g}$ trivially becomes an empty vector when the problem is not about locomotion.

Algorithm 1: Viterbi Algorithm With Particle Filtering.

1: $\delta_0(1:N) = 0$ $\triangleright$Initialization
2: **for** $k = 1, ..., K$ **do** $\triangleright$Recursion
3:      **for** $i = 1, ..., N_0$ **do**
4:          $\mathbf{g}_k^{(i)} \sim \mathcal{N}\left(\mathbf{g}_{k-1}^{(i)} + R_\theta\left(\mathbf{y}_{k-1}^{(i)}\right)_v, R_\theta \Sigma_{Y_v}(\mathbf{x}_{k-1}^{(i)}) R_\theta^T\right)$
5:          $\mathbf{x}_k^{(i)} \sim \mathcal{N}\left(\mu_X(\mathbf{x}_{k-1}^{(i)}), \Sigma_X(\mathbf{x}_{k-1}^{(i)})\right)$
6:          $\mathbf{y}_k^{(i)} = \mu_Y(\mathbf{x}_k^{(i)})$
7:          $[val, j^*] = \max_j \left[\delta_{t-1}(j) + \log p(\bar{\mathbf{x}}_k^{(i)} | \bar{\mathbf{x}}_{k-1}^{(j)})\right]$
8:          $\delta_k(i) = val - q_k(\mathbf{y}_k^{(i)}, \mathbf{g}_k^{(i)})$
9:          $\psi_k(i) = j^*$
10:         $w_k^{(i)} = w_{k-1}^{(i)} \exp(-q_k(\mathbf{y}_k^{(i)}, \mathbf{g}_k^{(i)}))$
11:      **end for**
12:      $w_k^{(i)} = w_k^{(i)} / \sum_j w_k^{(j)}, \forall i \in \{1, ..., N_0\}$
13:      Resample if $\left(\sum_i (w_k^{(i)})^2\right)^{-1} < N_0/2$
14: **end for**
15: $i_T^* = \text{argmax}_i \delta_K(i)$ $\triangleright$Termination
16: $\mathbf{y}_K^* = \mathbf{y}_K^{(i_T^*)}$
17: **for** $k = K-1, ..., 1$ **do** $\triangleright$Backtracking
18:      $i_k^* = \psi_{k+1}(i_{k+1}^*)$
19:      $\mathbf{y}_k^* = \mathbf{y}_k^{(i_k^*)}$
20: **end for**
21: **return** $\mathbf{y}_{1:K}^* = \{\mathbf{y}_1^*, \mathbf{y}_2^*, ..., \mathbf{y}_K^*\}$

---

algorithm recursively computes the MAP trajectory along the approximated discrete space.

The proposed algorithm, shown in Algorithm 1, is based on the DP procedure like the Viterbi algorithm, and consists of forward recursion and backtracking. In the forward recursion, the algorithm constructs the discrete state space by propagating particles of the PF and computes the optimal (partial) trajectories using the DP recursion up to the current time-step. If the final time is reached, it finds the most-likely final state and the backtracking procedure constructs the optimal trajectory from the chosen state in the backward direction. In detail, through the forward recursion, the augmented latent variables are propagated as in (4) and (17), and the corresponding pose is also realized (line 4–6). We restrict the noise in the GP realization (6) to 0, i.e., $\mathbf{y} = \mu_Y(\mathbf{x})$, because this noise only makes resulting motions rougher. However, this restriction can be easily removed if more complex poses are necessary for the planning. If we take the log to (20), the equation is simplified further with summation as:

$$\mathbf{y}_{1:K}^* = \{\mu_Y(\mathbf{x}_k^*)\}_{k=1,...,K},$$

$$\bar{\mathbf{x}}_{1:K}^* = \underset{\bar{\mathbf{x}}_{1:K}}{\text{argmax}} \sum_{k=1}^K \log p^o(o_k = 1 | \mu_Y(\mathbf{x}_k), \mathbf{g}_k) p(\bar{\mathbf{x}}_k | \bar{\mathbf{x}}_{k-1})$$

(21)

with constraints, $\mathbf{y}_k = \mu_Y(\mathbf{x}_k), \forall k \in \{1, ..., K\}$. Then the parent nodes that maximize the log-posteriors up to the current state are determined and the log-posteriors of the partial trajectories are computed (line 7–9). Note that in line 7, $\log p(\bar{\mathbf{x}}_k | \bar{\mathbf{x}}_{k-1})$ has a simple closed form because the dynamics of $\mathbf{x}$ and $\mathbf{g}$ follows the Gaussian distributions as in (4) and (17), respectively.

By computing the weights of particles in line 10, the algorithm can perform resampling procedure which helps the discrete approximation of the space to span valid regions in the original continuous space; the states having low posterior probabilities (or equivalently, having high costs in the planning problem) up to the current time-step tend not to be re-sampled (line 12–13). After the forward recursion, the algorithm picks the most likely final state $\mathbf{y}_K^*$ (line 15–16), and then constructs the whole trajectory with backtracking by looking at its ancestry (line 17–20).

D. Multiscale Acceleration With Path Integral Control

The proposed algorithm approximates the continuous state space at time k by using the PF up to that time. If only a small number of particles are used in the algorithm, they cannot expand the valid state space as shown in Fig. 5(a), where all particles failed to reach a goal region. Consequently, when a problem has a complex cost function (induced by environmental or task complexity), we cannot help but use a large number of particles to span the space effectively. Having a small number of particles, however, is crucial because the computational complexities of the PF and the DP procedures are $\mathcal{O}(N_0 K)$ and $\mathcal{O}(N_0^2 K)$, respectively. If particles are guided into the better region with useful heuristic, better sample efficiency can be achieved. This is also closely related to the idea of auxiliary PF [28], where the particles are propagated considering the future measurement. Trivially, the best sample efficiency can be achieved by propagating particles using the optimal control policy of the original OC problem [29], but it is, of course, unrealistic because we do not have the optimal control policy before solving the problem.

In this section, we introduce an efficient multiscale method that guides particles to better regions. With the discrete time-step h, $K = T/h$, and with slight abuse of notation, $q_T(\cdot) = q_K(\cdot)$, $q_k(\cdot) = q(\cdot, kh)$, the OC problem in (8)–(11) can be viewed as the discretized version of the following continuous-time OC problem:

$$d\mathbf{x}(t) = F(\mathbf{x}(t))dt + B(\mathbf{x}(t))(\mathbf{u}(t)dt + d\mathbf{w}(t)), \quad (22)$$

$$J = \mathbb{E}\left[q_T(\mathbf{x}(T)) + \int_0^T q(\mathbf{x}(t), t) + \frac{1}{2}||\mathbf{u}(t)||_2^2 dt\right], \quad (23)$$

because the KL-divergence term in (11) can be interpreted as the quadratic control cost, i.e., $D_{KL}(\pi(\cdot|\mathbf{x})||p(\cdot|\mathbf{x})) = \frac{h}{2}||\mathbf{u}||_2^2$.[2] The idea of the proposed method in this section is to sequentially solve approximate discretized problems with different time-steps $M_l \times h$ in the coarse to fine direction, where a solution of a coarser approximate problem is utilized as a priori of a finer approximate problem. Because the discrete time-length is shorter in the coarser approximate problem, a larger number of particles can be used while maintaining the computational complexity. We adopt the path integral control method to obtain the optimal control $\mathbf{u}^*$ which also can be solved efficiently within the *"optimal control via inference"* perspective.

Let the original OC problem be level 0 and $M_l \geq 1$ be the number of time steps aggregated in level l for $l = 1, ..., L$. That is, lth level OC problem is the discretized version of

---

[2]Note that the KL-divergence between Gaussian distributions $\mathcal{N}(\mu, \Sigma)$ having difference μ but same Σ is given by $\frac{1}{2}(\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2)$ and, in the case of (22), $(\mu_1 - \mu_2) = B(\mathbf{x})\mathbf{u}h$ and $\Sigma = B(\mathbf{x})B(\mathbf{x})^T h$, which results in this interpretation.

Algorithm 2: PI Control With PF at Level l.

1: Input: approximate solution at $(l+1)$ level problem, $\mathbf{u}^{l+1}$.
2: Create $\bar{\mathbf{u}}^{l+1}$ by up-sampling $\mathbf{u}^{l+1}$ by a factor of M_l.
3: **for** $k = 1, ..., K_l$ **do**
4:     **for** $i = 1, ..., N_l$ **do**
5:         $\mathbf{g}_k^{(i)} \sim \mathcal{N}\left(\mathbf{g}_{k-1}^{(i)} + M_l R_\theta \left(\mathbf{y}_{k-1}^{(i)}\right)_v, M_l R_\theta \Sigma_{Y_v}$
        $(\mathbf{x}_{k-1}^{(i)}) R_\theta^T\right)$
6:         $\mathbf{w}_{k-1}^{(i)} \sim \mathcal{N}(0, I_d)$
7:         $\mathbf{x}_k^{(i)} = \mu_X^l(\mathbf{x}_{k-1}^{(i)}, \bar{\mathbf{u}}_k^{l+1}) + (\Sigma_X^l(\mathbf{x}_{k-1}^{(i)}))^{1/2}\mathbf{w}_{k-1}^{(i)}$
8:         Append $\{\mathbf{w}_{k-1}^{(i)}, \{\mathbf{x}_k^{(i)}, \mathbf{g}_k^{(i)}\}\}$ into $\{W_{0:k-1}^{(i)},$
        $\bar{X}_{1:k}^{(i)}\}$
9:         $\mathbf{y}_k^{(i)} = \mu_Y(\mathbf{x}_k^{(i)})$
10:        $w_k^{(i)} = w_{k-1}^{(i)} \exp(-M_l q_k(\mathbf{y}_k^{(i)}, \mathbf{g}_k^{(i)}))$
11:     **end for**
12:    $w_k^{(i)} = w_k^{(i)} / \sum_j w_k^{(j)}, \forall i \in \{1, ..., N\}$
13:    Resample $\{w_k^{(i)}, W_{1:k}^{(i)}, \bar{X}_{1:k}^{(i)}\}$ if $\left(\sum_i (w_k^{(i)})^2\right)^{-1}$
    $< N/2$
14: **end for**
15: $\bar{\mathbf{u}}_{0:K_l-1}^l \leftarrow \{\mathbf{u}_0^l, \mathbf{u}_1^l, ..., \mathbf{u}_{K_l-1}^l\}$ ▷Equation (25)
16: **return** $\mathbf{u}_{0:K-1}^l \leftarrow \bar{\mathbf{u}}_{0:K_l-1}^l \otimes \mathbf{1}_{M_l}$

---

the continuous-time OC problem (22)–(23) with time-step $h \times M_l$. The lth level dynamics is guided by the $(l+1)$th level optimal control $\mathbf{u}_k^{l+1}$ and therefore is given by $\mathbf{x}_{k+1}^l = \mu_X^l(\mathbf{x}_k^l, \mathbf{u}_k^{l+1}) + (\Sigma_X^l(\mathbf{x}_k^l))^{1/2}\mathbf{w}_k, \forall k = 1, ..., K_l - 1$, where

$$\mu_X^l(\mathbf{x}, \mathbf{u}) = \mathbf{x} + M_l(\mathbf{x} - \mu_X(\mathbf{x})) + M_l \Sigma_X^{1/2}(\mathbf{x})\mathbf{u}h,$$

$$\Sigma_X^l(\mathbf{x}) = M_l\left(k_X(\mathbf{x}, \mathbf{x}) - \mathbf{k}_X(\mathbf{x})^T \mathbf{K}_X^{-1}\mathbf{k}_X(\mathbf{x})\right), \quad (24)$$

and $K_l = T/(hM_l) = K/M_l$ is a discrete time-length. Suppose a set of N_l simulated trajectories and weights $\{w_k^{(i)}, X_{1:k}^{(i)}\}_{i=1,...,N_l}$ is obtained from the PF, which approximates the optimally controlled trajectory distribution. The path integral control algorithm computes the optimal control solution that minimizes the KL divergence between the controlled trajectory distribution and the optimal one with moment matching [30], and the optimal control is given as:

$$\mathbf{u}_k^l \approx \mathbf{u}_k^{l+1} + \frac{1}{hM_lN_l}\sum_{i=1}^{N_l} w_{K_l}^{(i)}\sqrt{M_l}\mathbf{w}_k^{(i)}. \quad (25)$$

Such a procedure at level l is summarized in Algorithm 2. First, the approximate solution at the upper level $\mathbf{u}^{l+1}$ is up-sampled by a factor of M_l (line 2), where $\mathbf{u}_k^{L+1} = 0$ at the coarsest level. Guided by this, the algorithm propagates particles (line 5–7), realizes poses of robot (line 9), and evaluates particles' weights (line 10, 12). The algorithm stores the particles' coordinates and the noises that propagated particles in the form of a trajectory, and re-samples them when necessary (line 8 and 13, respectively). After the filtering procedure, the approximate optimal control is computed as (25), stretches it back by a factor of M_l and returns it (line 15–16). With this procedure, the optimal control is sequentially computed from the coarsest level, $l = L$, to

the finest level, $l = 1$. Then finally, the finest solution will be utilized to guide the particle propagation of Algorithm 1; that is, in the line 6 of Algorithm 1, the mean of the particle propagation is changed into $\mathbf{x}_k^{(i)} \sim \mathcal{N}(\mu_X(\mathbf{x}_{k-1}^{(i)}) + \mathbf{u}_{k-1}^1 h, \Sigma_X(\mathbf{x}_{k-1}^{(i)}))$. This guidance is expected to lead that only a small number of particles are enough to expand the valid state space.

IV. EXAMPLE: HUMANOID LOCOMOTION PLANNING

A. Training Data and Hyperparameter Settings

As an illustrative example, we specifically consider a humanoid robot with 56-dimensional configuration space. The Carnegie Mellon University motion capture (CMU mocap) database is used to learn the latent variable model. The 56-dimensional configurations consist of angles of all joints, roll and pitch angles, vertical position of center of the spine (the root), yaw rate of the root, and horizontal velocity of the root. The original data were written at 120 Hz, but we down-sampled them into 30 Hz to decrease the size of the observation set while maintaining the quality of motions for capturing the valid temporal information.

For priors of the hyperparameters, we utilized the commonly-used radial basis function (RBF) for k_X and k_Y, i.e., $k_X(\mathbf{x}, \mathbf{x}') = \alpha_1 \exp(-\frac{\alpha_2}{2}||\mathbf{x} - \mathbf{x}'||^2) + \alpha_3^{-1}\delta_{\mathbf{x},\mathbf{x}'}$, and $k_Y(\mathbf{x}, \mathbf{x}') = \beta_1 \exp(-\frac{\beta_2}{2}||\mathbf{x} - \mathbf{x}'||^2) + \beta_3^{-1}\delta_{\mathbf{x},\mathbf{x}'}$, where δ is a Kronecker delta function. As is widely used, we used the uninformative prior on the kernel hyperparameters as $p(\alpha) \propto \prod_i \alpha_i^{-1}$, $p(\beta) \propto \prod_i \beta_i^{-1}$, which has shown effective regularizations [6], [7]. We also used the back-constraints introduced in [8]: because the locomotion has some periodicity, the corresponding latent variable model also does. First, we extracted the phase of the motion ϕ and augmented it to the observation $\mathbf{Y}$. Then, the back-constraints were used such that the last two latent dimensions had periodic structures as: $x_{n,d-1} = \sum_{m=1}^N a_m^{\cos}k_{rbf}(\cos(\phi_n), \cos(\phi_m)) + a_0^{\cos}\delta_{n,m}$, and $x_{n,d} = \sum_{m=1}^N a_m^{\sin}k_{rbf}(\sin(\phi_n), \sin(\phi_m)) + a_0^{\sin}\delta_{n,m}$. Moreover, the standard RBF back constraints were used in other dimensions. The GPmat toolbox [31] is utilized to learn the latent variable models.

B. Results

For the first numerical experiment, we trained the model using the data sets for walking, fast walking and jogging. The latent space was set to be 3-D, where the first and the last two dimensions were initialized as being associated with the forward velocity of the root and the phases, respectively. Note that the optimization problem of GP-LVMs is non-convex and the learning algorithm is based on the gradient descent method [6], so proper initialization is crucial. The task for the first experiment was to move forward without stepping on the red lines shown in Fig. 2. We set the cost function to create a penalty for the deviation from a desired heading angle, $\theta_d = 0$, a desired x position, $x_d = 0$, forward velocity, $v_d = 5$ m/s, and when the foot touched the red lines, i.e., $q(\mathbf{y}, \mathbf{g}) = q_{\text{obs}}(\mathbf{y}, \mathbf{g}) + \theta^2 + 0.01|x| + 0.1(v - 5)^2$. The foot positions were computed through the forward kinematics:

$$q_{obs}(\mathbf{y}, \mathbf{g}) = \begin{cases} \infty, & \text{if } FK_{foot}(\mathbf{y}, \mathbf{g}) \in D_{redline}, \\ 0, & \text{otherwise,} \end{cases}$$
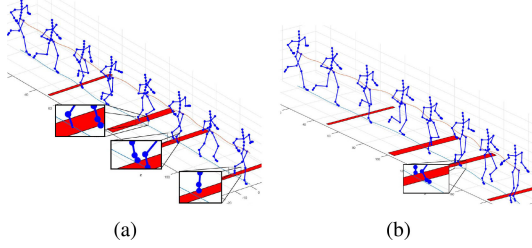
Fig. 2. Resulting trajectories for the experiment 1. The MAP trajectory with the cost function (a) without and (b) with collision checking.

where $FK_{foot}(\cdot)$ is the function of the forward kinematics for the foot joints. 500 particles were used in the Algorithm 1 with time horizon $K = 90$ (i.e., $T_f = 3$ sec.). In addition, we ignored the randomness in the last 2 dimensions of latent space (which are for phase), i.e., only the learned mean dynamics is used for phase as $\mathbf{x}_{k+1} = \mu_X(\mathbf{x}_k)$, because we found that the noise in those dimensions made the resulting motion too jerky and was not helpful for the planning. Fig. 2(a) and (b) show the MAP trajectory without and with collision checking, respectively. It is shown that the proposed algorithm could find the natural movements that step over the forbidden regions while the motion from the passive dynamics steps on the red lines. In addition, even though our training data consisted of around 3–4 cycles for each motion, the learned generative model produced natural longer movements continuously.

For the second experiment, we added the datasets for left and right turns to see the trajectory make detours around obstacles. The latent space was set to be 4-D here, where the first two dimensions were initialized as corresponding to the yaw rate and forward velocity of the root, while the last two dimensions were for the phases, as in the first example. The cost function penalized the situations when the robot collided with any obstacle, or left the domain, or reached positions too the distant from the goal region, i.e., $q(\mathbf{y}, \mathbf{g}) = q_{obs}(\mathbf{y}, \mathbf{g}) + q_{bnd}(\mathbf{g}) + 10^{-5}q_{goal}(\mathbf{g})$, where $q_{obs}(\mathbf{y}, \mathbf{g})$ and $q_{bnd}(\mathbf{g})$ are ∞ if $FK(\mathbf{y}, \mathbf{g}) \in D_{obstacle}$ and $\mathbf{g} \notin \mathcal{D}$, receptively, and are 0 otherwise; $q_{goal}(\mathbf{g})$ is the square of the distance to the goal region, computed using the FMT* algorithm [2]. The domain $\mathcal{D}$ and the goal region were set to be $[-45, 45] \times [0, 250]$ and around $(30, 230)$, respectively. 1000 particles were used in the Algorithm 1 with time horizon $K = 450, 150$ (i.e., $T_f = 15, 5$ sec.) for walking and running tasks, respectively. We ran our algorithm for the cases of various initial positions and orientations and the resulting motion trajectories are shown in Fig. 3. It is shown that the proposed algorithm was able to generate smooth and natural motion sequences toward the goal region without collision. Fig. 4 shows the learned 4-D latent space constructed by various motions. It is observed that, while varying from the initial guess (in Fig. 4(a) and (c)), the motions are well-organized in the latent space (in Fig. 4(b) and (d)). We observed that the learned model separated the walking and running motions into two clusters and the transition motion from stand to running was embedded as the small "bridge" between them. As a result, the learned generative model hardly produced the transition between walking and running; we expect that more various transition motions are necessary to make more flexible models. It can be also seen from the supplementary video that
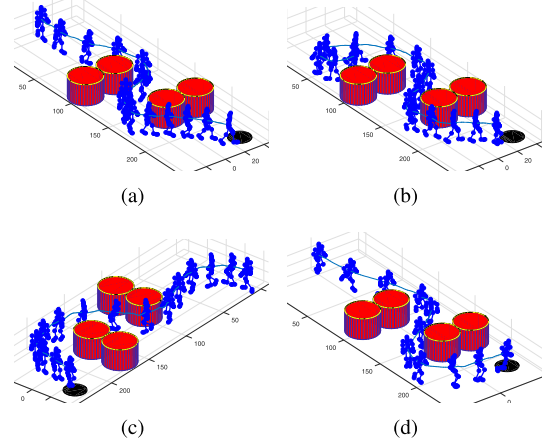

Fig. 3. Resulting trajectories for the experiment 2 from various initial configurations. The goal region is marked by the black circle. The poses are drawn for every (a)–(c) 30 and (d) 15 steps, i.e., 1 and 0.5 second.
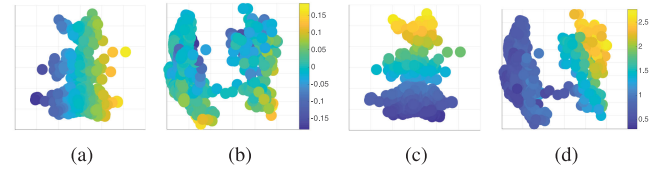

Fig. 4. Representation of various motions in the 4-D latent space projected to the (x_1, x_2) subspace. The points are colored by (a), (b) yaw rate and by (c), (d) forward velocity of the Mocap data. (a) Initial. (b) Learned. (c) Initial. (d) Learned.
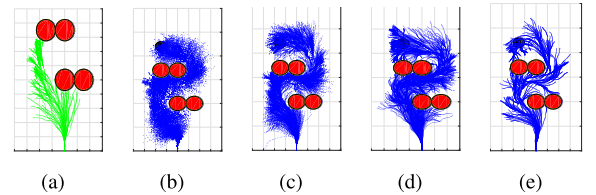

Fig. 5. The particles from (a) the naive PF approach with 50 particles and from the recursive multiscale procedure with (b) $M_3 = 8$, $N_3 = 800$, (c) $M_2 = 4$, $N_2 = 400$, (d) $M_1 = 2$, $N_1 = 200$, (e) $M_0 = 1$, $N_0 = 50$.

the initialized latent variable model cannot generate proper natural motions. We would refer the readers to the supplementary video for more visualization.

Finally, we tested multiscale acceleration method. The multiscale parameters were set as $N_l = 200, 400, 800$ and $M_l = 2, 4, 8$ and $N_0 = 50$ particles are used in the original scale. These multiscale parameters should be chosen carefully, because the approximate solution (25) may fail to guide the finer level dynamics properly if a gap between the scales is too large. Fig. 5(b)–(e) depict the particles from the proposed recursive procedure. It is observed that as the approximation level decreases, smaller number of particles are well-guided to the goal region; the larger number of particles successfully expanded the state space at the higher level, and then only 50 particles succeeded to be propagated toward the goal at the original scale. We also compared the performances of Algorithm 1 with various

TABLE I
COMPUTATIONAL COMPLEXITY AND SUCCESS RATE

Case	PF	DP	CPU (sec)	Env. 1	Env. 2	Env. 3
$N_0 = 50$	1	1	13.25	100%	23%	9%
$N_0 = 500$	10	100	140.16	100%	82%	77%
$N_0 = 1000$	20	400	262.46	100%	98%	95%
Multi-Scale	7	1	87.17	100%	85%	85%

numbers of particles ($N_0 = 50, 500, 1000$) in various environments having different obstacles: the environment 3 is same as that of Fig. 3(a), the environment 2 only had 2 obstacles close to the goal region, and the environment 1 had no obstacles; the initial state and the goal region were same as those of Fig. 3(a). The comparison result is shown in Table I. The second, third and fourth columns denote the relative computational complexities of PF, i.e., $\mathcal{O}(N_0 K)$, DP, i.e., $\mathcal{O}(N_0^2 K)$, and computation time for whole planning procedures, respectively. Note that, for the algorithm with the multiscale acceleration, the computational complexity of DP is same to the case of $N_0 = 50$ because the same number of particles are used, and the computational complexity of PF is 7 times larger than the case of $N_0 = 50$ because those are twice at each level $N_l/M_l = 100, \forall l = 1, 2, 3$ and same at the original level. We found that most computational budgets were spent by GP realizations (around 50%) and forward kinematics for collision checking (around 30%), which is proportional to the number of data N, particles N_0, and the planning horizon K. Therefore, the planing algorithm can be accelerated further by using ideas of sparse GP-LVMs [6] and more efficient forward kinematics and collision checking methods; we leave it as future works. The last three columns show the rate of each algorithm that at least one of particles achieved the goal region. Table I shows that though a larger number of particles need to be used as the environment becomes complex, which results in huge computational complexity, the proposed multiscale method can guarantee the performance while maintaining the computational complexity properly.

V. CONCLUSIONS

In this work, we proposed an efficient framework for generating the motion trajectory of a robot with high degrees of freedom. The framework included a probabilistic generative model for the motion sequence from demonstration data using the GP-LVM method. The constructed low-dimensional model was then combined with the probabilistic model of the OC problem. Finally, we proposed an efficient approximate MAP trajectory estimation algorithm modified to utilize the dynamic programming procedure and the multiscale path integral control method to increase the sample efficiency.

REFERENCES

[1] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.
[2] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *Int. J. Robot. Res.*, pp. 883–921, 2015.
[3] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," *Proc. IEEE Amer. Control Conf.*, 2005, pp. 300–306.
[4] M. Toussaint, "Robot trajectory optimization using approximate inference," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 1049–1056.
[5] P. Vernaza and D. D. Lee, "Learning and exploiting low-dimensional structure for efficient holonomic motion planning in high-dimensional spaces," *Int. J. Robot. Res.*, vol. 31, no. 14, pp. 1739–1760, 2012.
[6] N. Lawrence, "Probabilistic non-linear principal component analysis with Gaussian process latent variable models," *J. Mach. Learn. Res.*, vol. 6, pp. 1783–1816, 2005.
[7] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Gaussian process dynamical models for human motion," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 2, pp. 283–298, 2008.
[8] R. Urtasun, D. J. Fleet, A. Geiger, J. Popović, T. J. Darrell, and N. D. Lawrence, "Topologically-constrained latent variable models," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 1080–1087.
[9] S. Levine, J. M. Wang, A. Haraux, Z. Popović, and V. Koltun, "Continuous character control with low-dimensional embeddings," *ACM Trans. Graph.*, vol. 31, no. 4, 2012, Art. no. 28.
[10] R. Urtasun, D. J. Fleet, and P. Fua, "3D people tracking with Gaussian process dynamical models," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2006, vol. 1, pp. 238–245.
[11] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2014.
[12] N. Chen, M. Karl, and P. van der Smagt, "Dynamic movement primitives in latent space of time-dependent variational autoencoders," in *Proc. IEEE-RAS 16th Int. Conf. Humanoid Robots (Humanoids)*, 2016, pp. 629–636.
[13] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018.
[14] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 2786–2793.
[15] J.-S. Ha, Y.-J. Park, H.-J. Chae, S.-S. Park, and H.-L. Choi, "Adaptive path-integral approach for representation learning and planning," *Int. Conf. Learn. Representations Workshop*, 2018.
[16] J.-S. Ha, Y.-J. Park, H.-J. Chae, S.-S. Park, and H.-L. Choi, "Adaptive path-integral approach to representation learning and planning for dynamical systems," arXiv:1807.02128, 2018.
[17] E. Todorov, "General duality between optimal control and estimation," in *Proc. IEEE Conf. Decis. Control*, 2008, pp. 4286–4292.
[18] K. Rawlik, M. Toussaint, and S. Vijayakumar, "On stochastic optimal control and reinforcement learning by approximate inference," in *Proc. Robot. Sci. Syst.*, 2012, pp. 3052–3056.
[19] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *J. Mach. Learn. Res.*, vol. 11, pp. 3137–3181, 2010.
[20] E. Todorov, "Finding the most likely trajectories of optimally-controlled stochastic systems," *IFAC Proc. Volumes*, vol. 44, no. 1, pp. 4728–4734, 2011.
[21] P. Hämäläinen, J. Rajamäki, and C. K. Liu, "Online control of simulated humanoids using particle belief propagation," *ACM Trans. Graph.*, vol. 34, no. 4, 2015, Art. no. 81.
[22] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time Gaussian process motion planning via probabilistic inference," arXiv:1707.07383, 2017.
[23] J.-S. Ha and H.-L. Choi, "A topology-guided path integral approach for stochastic optimal control," in *Proc. IEEE Int. Conf. Robot. Automat.*, pp. 4605–4612, 2016.
[24] N. D. Lawrence and J. Quiñonero-Candela, "Local distance preservation in the GP-LVM through back constraints," in *Proc. 23rd Int. Conf. Mach. Learn.*, pp. 513–520. ACM, 2006.
[25] E. Todorov, "Efficient computation of optimal actions," *Proc. Nat. Acad. Sci.*, vol. 106, no. 28, pp. 11478–11483, 2009.
[26] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. Cambridge, MA, USA: MIT Press, 2009.
[27] S. Godsill, A. Doucet, and M. West, "Maximum a posteriori sequence estimation using Monte Carlo particle filters," *Ann. Inst. Statist. Math.*, vol. 53, no. 1, pp. 82–96, 2001.
[28] M. K. Pitt and N. Shephard, "Filtering via simulation: Auxiliary particle filters," *J. Amer. Statist. Assoc.*, vol. 94, no. 446, pp. 590–599, 1999.
[29] S. Thijssen and H. Kappen, "Path integral control and state-dependent feedback," *Phys. Rev. E*, vol. 91, no. 3, 2015, Art. no. 032104.
[30] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2016, pp. 1433–1440.
[31] N. D. Lawrence *et al.*, "GPmat.", 28th May 2015 [Online]. Available: https://github.com/SheffieldML/GPmat