Abstract
The objective function used in trajectory optimization is often non-convex and can have an infinite set of local optima. In such cases, there are diverse solutions to perform a given task. Although there are a few methods to find multiple solutions for motion planning, they are limited to generating a finite set of solutions. To address this issue, we present an optimization method that learns an infinite set of solutions in trajectory optimization. In our framework, diverse solutions are obtained by learning latent representations of solutions. Our approach can be interpreted as training a deep generative model of collision-free trajectories for motion planning. The experimental results indicate that the trained model represents an infinite set of homotopic solutions for motion planning problems.

Keywords
Motion planning, Optimization, Learning latent representations

1 Introduction
Motion planning has been investigated for decades in the field of robotics because it is an essential component in many robotic systems. A popular approach for motion planning is optimization-based methods, which finds a solution that minimizes the objective function. Various optimization methods have been leveraged in motion planning in robotics (Khatib 1986; Zucker et al. 2013; Schulman et al. 2014). Ideally, an objective function should be designed such that the solution is unique and the optimization problem can be solved stably. If we can formulate our problem as a convex optimization, we can leverage the techniques established in previous studies (Boyd and Vandenberghe 2004). However, the objective function in motion planning is often non-convex and may have many local optima. To circumvent the difficulty of optimizing the trajectory with respect to the non-convex objective function, prior work has focused on robustly finding a solution. As previous studies have established various methods for robustly finding a single solution in motion planning, we address an unexplored aspect of motion planning, that is, finding diverse solutions.

Two trajectories are called homotopic when one can be continuously deformed into another (Jaillet and Simeon 2008; Hatcher 2002). Previous studies by Jaillet and Simeon (2008); Orthey et al. (2020) indicated that there may be an infinite set of homotopic solutions in motion planning. Figure 1 shows an example of homotopic collision-free paths in motion planning. Paths $\xi_1$ and $\xi_2$ are homotopic, and both are collision-free. Although only two paths are visualized in Figure 1, there is an infinite set of collision-free paths between $\xi_1$ and $\xi_2$. Existing motion planning methods often use a generic objective function, which is applicable for various tasks but often has many solutions. It is possible to specify a unique solution by posing additional constraints, but this is impractical for users who are not experts on optimization and/or robotics. For this reason, it is preferable to suggest various solutions and allow the users to select the preferable one among the candidates. However, existing methods such as a method proposed by Osa (2020) are limited to generating a fixed number of solutions, and it is not straightforward to generate a new “middle” solution out of the two obtained solutions.

Our contribution is to propose a framework that can generate diverse collision-free paths in motion planning. In Figure 1(b), $z$ represents the latent variable learned by the proposed method, and various solutions can be generated by changing the values of $z$. In this study, we present a practical algorithm for capturing diverse solutions in motion planning.

Figure 1. Example of homotopic solutions in motion planning.
We first propose an optimization method that learns an infinite set of solutions for optimization. Our approach can be interpreted as training a deep generative model of optimal points in optimization. In the field of information geometry, an $n$-dimensional manifold is defined as a set of points such that each point has $n$-dimensional extensions in its neighborhood, and a point in a manifold can be specified using its coordinates (Amari 2016). Our framework encodes the diversity of the solutions into a continuous latent variable, and each solution has extensions in its neighborhood. In addition, each solution can be specified using the value of latent variable. In this context, our method learns the manifold of solutions in optimization, and the learned latent variable can be regarded as the coordinates of the obtained manifold. Therefore, we refer to our optimization method as Learning the Solution Manifold in Optimization (LSMO).

In this paper, we first present the derivation of LSMO. We then present a motion planning algorithm based on LSMO, which we refer to as Motion Planning by Learning the Solution Manifold (MPSM), and describe how we can adapt LSMO to solve the motion planning problems efficiently. In MPSM, a neural network that takes in a latent variable is trained to generate diverse collision-free trajectories. Using our approach, the variation of collision-free trajectories are encoded in a low-dimensional latent variable, and a user can intuitively obtain various collision-free trajectories in motion planning by changing the value of the latent variable. In experiments, to analyze the behavior of LSMO, we first applied it to tasks of maximizing test functions that have an infinite set of optimal points. We then evaluate MPSM on motion-planning tasks for a robotic manipulator. We empirically showed that MPSM can learn an infinite set of homotopic trajectories in motion-planning problems.

### 2 Related Work

#### 2.1 Motion planning methods in robotics

A popular class of motion planning methods is optimization-based methods, and Covariant Hamiltonian Optimization for Motion Planning (CHOMP) (Zucker et al. 2013), STOMP (Kalakrishnan et al. 2011), TrajOpt (Schulman et al. 2014), and Gaussian Process Motion Planner (GPMP) (Mukadam et al. 2018) are included in this class. These methods determine the trajectory that minimizes the objective function, which quantifies the quality of the trajectory. Sampling-based methods are also popular in motion planning for robotic systems. Probabilistic RoadMap (PRM) (Kavraki et al. 1996, 1998), Rapidly-exploring Random Trees (RRT) (LaValle and Kuffner 2001), RRT* (Karaman and Frazzoli 2011), BIT* (Gammell et al. 2020) are categorized as sampling-based methods. In sampling-based methods, collision-free configurations are sampled in a stochastic manner, and a path is planned by connecting the sampled configurations. Prior work showed that sampling-based methods work well for complex motion planning problems such as maze tasks. In addition, previous studies by Koert et al. (2016); Osa et al. (2017); Rana et al. (2017); M. A. Rana and Ratliff (2020); Mukadam et al. (2020) incorporated the learning-from-demonstration approach (Argall et al. 2009; Osa et al. 2018) with optimization-based and sampling-based approaches. Recent studies proposed methods that leverage the capability of neural networks (Srinivas et al. 2018; Jurgenson and Tamar 2019; Chen et al. 2020). These deep-learning-based methods are often built on sampling-based or optimization-based methods, and they exploit the generalization ability of neural networks to achieve robust and/or computationally efficient motion planning. Although the focus of this study is the optimization-based method, our work will also contribute to other types of motion planning methods because the various types of motion-planning methods interactively influence the advances in the field of motion planning.

#### 2.2 Finding Multiple Solutions in Motion planning

Works by Jaillet and Simeon (2008); Orthey et al. (2020) discussed homotopy and the deformability of trajectories, and they implied that there may be an infinite set of collision-free trajectories in motion planning. Figure 2 shows an example of such motion planning tasks. However, existing methods often ignore the existence of multiple solutions and find only a single solution. As a consequence, a motion planner may generate a solution that is different from the one that the user expected. If the user wishes to obtain another type of solution, then she/he needs to modify the objective function of motion planning or re-run the motion planner.
with different random seeds until a preferable solution is obtained. Recent studies by Toussaint et al. (2018, 2020) addressed manipulation planning by formulating a problem as a Logic-Geometric Program (LGP) (Toussaint 2015). Their method finds diverse plans to achieve a physical manipulation task using a tree-search-based algorithm. The study by Orthey et al. (2020) employed a tree-based architecture to represent multiple local minima in motion planning. Osa (2020) also recently proposed a motion planning algorithm that finds multiple solutions based on multimodal optimization. Although these methods allow the user to examine diverse solutions and select a preferable one among a given set of solutions, they are limited to generating a finite number of solutions. If the preferable solution is not included in the given set, then the user will need to rerun the motion planner with different random seeds as in other motion planning methods. In this work, we propose a framework that models an infinite set of solutions for motion planning. As our framework learns the continuous latent representation of solutions, the user will be able to examine diverse solutions intuitively and tune the type of solutions efficiently.

2.3 Multimodal optimization with black-box optimization methods

Prior studies (Goldberg and Richardson 1987; Deb and Saha 2010; Stoean et al. 2010; Agrawal et al. 2014; Karasawa et al. 2020) addressed multimodal optimization using black-box optimization methods, such as CMA-ES (Hansen and Ostermeier 1996) and the cross-entropy method (CEM) (de Boer et al. 2005). For example, a study by Agrawal et al. (2014) showed that control policies to achieve diverse behaviors can be learned by maximizing the objective function that encodes the diversity of solutions. Although these black-box optimization methods are applicable to a wide range of problems, it is difficult to apply them to the optimization of high-dimensional parameters. Motion planning tasks usually involve high-dimensional parameters. For example, if the manipulator has seven degrees of freedom (DoFs) and a trajectory is represented by 50 time steps, then the trajectory of the manipulator is represented by a vector with 350 dimensions. For this reason, it is not trivial to directly apply black-box optimization methods to motion planning tasks in robotics. In addition, the limitation of black-box multimodal optimization methods is that they can learn only a finite set of solutions.

2.4 Latent Representations in Reinforcement learning and imitation learning

Recent studies on imitation learning proposed methods for modeling diverse behaviors by learning latent representations (Li et al. 2017; Merel et al. 2019; Sharma et al. 2019). For example, Merel et al. (2019) proposed a method for learning diverse behaviors by learning continuous latent variables. Likewise, in the field of reinforcement learning (RL), previous studies proposed methods for learning the diverse behaviors (Eysenbach et al. 2019). Hierarchical RL methods (Bacon et al. 2017; Florensa et al. 2017; Vezhnevets et al. 2017; Osa et al. 2019) often learn a hierarchical policy given by \( \pi(a|s) = \sum_o \pi(o|s)\pi(a|s, o) \), where \( s \), \( a \), and \( o \) denote the state, action, and option, respectively. These methods can be interpreted as approaches that models diverse behaviors with a policy conditioned on the latent variable. Recent studies (Nachum et al. 2018, 2019; Schaul et al. 2015) investigated goal-conditioned policies \( \pi(a|s, g) \), where \( g \) denotes the goal. A goal-conditioned policy can also be viewed as a way of modeling diverse behaviors conditioned on a latent variable that has semantic meaning. However, the problem setting of RL is different from ours because our problem formulation does not involve the Markov decision process.

3 Problem Formulation

We denote by \( q_t \in \mathbb{R}^D \) the configuration of a robot manipulator with \( D \) degrees of freedom (DoFs) at time \( t \). Given the start configuration \( q_0 \) and the goal configuration \( q_T \), the task is to plan a smooth and collision-free trajectory \( \xi = [q_0, \ldots, q_T] \in \mathbb{R}^{D \times T} \), which is given by a sequence of robot configurations. This problem can be formulated as an optimization problem

\[
\xi^* = \arg\max_{\xi} R(\xi)
\]

(1)

where \( R(\xi) \) is the score function that quantifies the quality of a trajectory \( \xi \). Although previous studies formulate the motion planning problem as minimization of the cost function \( C(\xi) \), we employ the formulation in (1) to make the following discussion concise. In the proposed algorithm, we train the model \( p_\theta(\xi|z) \) to represent a distribution of optimal points.

In this study, we are particularly interested in problems where there exist multiple, and possibly infinite solutions. For example, the objective function shown in Figure 3(a) has an infinite number of solutions. The goal of our study is to learn the solution manifold when optimizing such objective functions. Instead of finding a single solution, we aim to train a model that represents the distribution of optimal points \( p_\theta(\xi) \) parameterized with a vector \( \theta \) given by

\[
p_\theta(\xi) = \int p_\theta(\xi|z)p(z)dz,
\]

(2)

where \( z \) is the latent variable. We train the model \( p_\theta(x|z) \) by maximizing the surrogate objective function

\[
J(\theta) = \mathbb{E}_{\xi \sim p_\theta(\xi)}[f(R(\xi))],
\]

(3)

where \( f(\cdot) \) is monotonically increasing and \( f(x) \geq 0 \) for any \( x \in \mathbb{R} \). Because \( f(\cdot) \) is monotonically increasing, maximizing \( R(\xi) \) is equivalent to maximizing \( f(R(\xi)) \). Therefore, \( f(\cdot) \) can be interpreted as a shaping function. This shaping function \( f(\cdot) \) is used to derive the proposed algorithm in Section 4.2.

4 Learning Solution Manifold in Optimization

If a dataset of diverse solutions is available, we can employ standard machine-learning techniques, such as generative adversarial networks (GANs) (Goodfellow et al. 2014) and variational autoencoders (VAEs) (Kingma and Welling 2014)
to train a generative model of optimal solutions. However, obtaining such a dataset of diverse solutions is challenging in practice. Hence, we present an algorithm for training the generative model of optimal solutions using non-optimal samples obtained stochastically from a proposal distribution.

4.1 Overview of Proposed Optimization Algorithm

We first present an overview of our optimization algorithm, which can be applied to general optimization problems that are not limited to the motion planning problems. In this section, we consider the problem of maximizing the objective function

\[
J(\theta) = \mathbb{E}_{x \sim p_{\theta}(x)}[f(R(x))],
\]

where \( p_{\theta}(x) \) represent a model parameterized by a vector \( \theta \).

\[
p_{\theta}(x) = \int p_{\theta}(x|z)p(z)dz.
\]

Here, \( x \) represents a data point in general and is not necessarily a trajectory.

The main concept of the proposed algorithm is illustrated in Figure 3. We focused on optimization problems that involve an infinite set of optimal points, as shown in Figure 3(a). The aim of our algorithm is to train a generative model of the optimal points.

To this end, we used non-optimal samples to train the model, e.g., samples obtained from a uniform distribution, as shown in Figure 3(b). If we naively use the original VAE objective function for the training, then the resulting model will generate points that do not correspond to the optimal solutions, as shown in Figure 3(c). To address this issue, we introduce an importance weight based on \( f(R(x)) \). If we scale samples based on \( f(R(x)) \), then we can obtain a density whose modes correspond to the optimal solutions, as shown in Figure 3(d). In the proposed method, the generative model \( p_{\theta}(x|z) \) is trained as a part of VAE by maximizing the weighted log-likelihood with respect to samples obtained from a proposal distribution.

In Fig. 3(e), circles represent the output of the model trained with LSMO, and the color of the circle indicates the value of the latent variable. As shown in Fig. 3(e), the output of the trained model continuously changes by continuously changing the value of the latent variable. Because the value of the latent variable indicates the similarity of the solution, the user can intuitively go through various solutions using the model trained with LSMO.

When generating a solution from the trained model, the user specifies the value of the latent value \( z \) and generates a sample from the trained neural network. As the output of the neural network model is the result of inference, it may not correspond to the exact optimal point in the objective function. Therefore, the output of the neural network model is fine-tuned to obtain the exact solution if necessary. The proposed algorithm is summarized in Algorithm 1. In the next section, we present the derivation of the proposed method and show the relation between the objective function in (4) and the weighted log-likelihood in the next section.

Algorithm 1 Abstract of Learning the Solution Manifold in Optimization (LSMO)

**Input:** Objective function \( R(x) \), proposal distribution \( p_{prop}(x) \), shaping function \( f \)

**Training phase:**
1. Generate \( N \) synthetic samples \( \{x_i\}_{i=1}^N \) from the proposal distribution \( p_{prop}(x) \)
2. Evaluate the objective function \( R(x_i) \) and compute the weight \( f(R(x_i)) \) for \( i = 1, \ldots, N \)
3. Train \( p_{\theta}(x|z) \) by maximizing \( \mathcal{L}(\theta, \psi) \) in (12)

**Generation phase:**
4. Generate \( x^* \) with \( p_{\theta}(x|z) \) by specifying the value of \( z \)
5. (Optional) Fine-tune \( x^* \) with a gradient-based method

Return: \( x^* \)

4.2 Learning Latent Representations in Optimization

To derive our algorithm, we first consider the lower bound of the surrogate objective function \( J(\theta) \) in (4) in the same manner as in previous studies by Dayan and Hinton (1997); Kober and Peters (2011); Osa (2020). Although evaluating \( J(\theta) \) requires computing the expectation with respect to samples drawn from \( p_{\theta}(x) \), iteration of sampling from \( p_{\theta}(x) \) and updating \( \theta \) would be time-consuming, especially when \( p_{\theta}(x) \) is modeled with a neural network. Instead, we consider a proposal distribution \( p_{prop}(x) \) for generating a set of samples \( X = \{x_i\}_{i=1}^N \). To derive the relation between \( p_{prop}(x) \) and \( J(\theta) \), we apply Jensen’s inequality as follows:

\[
\log J(\theta) = \log \int p_{\theta}(x)f(R(x))dx
\]

\[= \log \int p_{prop}(x)f(R(x))\frac{p_{\theta}(x)}{p_{prop}(x)}dx\]

\[\geq \int p_{prop}(x)f(R(x))\log \frac{p_{\theta}(x)}{p_{prop}(x)}dx\]

\[= \int p_{prop}(x)f(R(x))\log p_{\theta}(x)dx + \text{const.} \quad (6)
\]

In the derivation of the lower bound in (6), we used \( f(R(x)) > 0 \) for any \( x \) from line 2 to line 3. Based on the lower bound in (6), given a set of data points \( X = \{x_i\}_{i=1}^N \) drawn from the proposal distribution \( p_{prop}(x) \), maximizing the weighted log-likelihood

\[L(\theta; X) = \int f(R(x))p_{prop}(x)\log p_{\theta}(x)dx \quad (7)\]

\[\approx \frac{1}{N} \sum_{i=1}^{N} f(R(x_i))\log p_{\theta}(x_i), \quad (8)\]

is equivalent to maximizing the lower bound of the objective function \( J(\theta) \) in (3).

To train the model \( p_{\theta}(x|z) \), we also leverage the variational lower bound as in VAE (Kingma and Welling 2014). The variational lower bound on the marginal likelihood of data point \( i \) is given by

\[
\log p_{\theta}(x_i) \geq \mathcal{L}(\psi; \theta; x_i)
\]

\[= -D_{KL}(q_{\psi}(z|x_i)||p(z)) + \mathbb{E}_{z \sim q(z|x_i)}[\log p_{\theta}(x_i|z)] \quad (9)
\]
Using the variational lower bound in (9), the lower bound of the objective function in (8) is given by

\[ L(\theta; X) \geq \mathcal{L}(\psi, \theta; X) \]

where \( \ell(\theta, \psi) \) is given by

\[ \ell(\theta, \psi) = \log p_\theta(x|z) - \gamma D_{KL}(q_\psi(z|x)||p(z)) - C_z. \]

where \( C_z \) is the information capacity of latent variable \( z \), and \( \gamma \) is a coefficient. The study by Dupont (2018) showed that this objective function encourages learning disentangled latent representations.

Although we showed how to maximize the lower bound of the surrogate objective function \( J(\theta) \), one needs to be aware that our approach is based on amortized variational inference (Cremer et al. 2018) because training of the neural network is based on that of VAE. In other words, our training procedure is amortized over the dataset instead of optimizing the output for each data point. Therefore, the output of the trained model \( p_\theta(x|z) \) may not be the exact solution to the optimization problem. Thus, it may be necessary to fine-tune the output of \( p_\theta(x|z) \) in practice, e.g., using a gradient-based method. We use the shaping function \( f(R(x)) = \exp(\alpha R(x)) \) in our implementation, and we deal with a scaling parameter \( \alpha \) as a hyperparameter. We investigate the effect of the value of \( \alpha \) in the experiments described in Section 7.

### 4.2.1 Connection to Density Estimation

To describe the connection to the density estimation, we consider a distribution

\[ p_{\text{target}}(x) = \frac{f(R(x))}{Z}, \]

which we refer to as the target distribution and \( Z \) is a partition function given by \( Z = \int f(R(x)) \, dx \). When \( p_{\text{target}}(x) \) is followed, a sample with a higher score is drawn with a higher probability. The problem of estimating the density induced by the target distribution \( p_{\text{target}}(x) \) can be formulated as

\[ \min_{\theta} D_{KL}(p_{\text{target}}(x)||p_\theta(x)), \]

where \( D_{KL}(p_{\text{target}}(x)||p_\theta(x)) \) is the KL divergence. Given a set of samples \( X = \{x_i\}_{i=1}^N \) drawn from the proposal
distribution, the minimizer of $D_{KL}(p_{\text{target}}(x)||p_{\theta}(s, \tau))$ is given by the maximizer of the weighted log likelihood:

$$L'(\theta) = \int W(x)p_{\text{prop}}(x) \log p_{\theta}(x)dx$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} W(x_i) \log p_{\theta}(x_i),$$

where $W(x)$ is the importance weight given by

$$W(x) = \frac{p_{\text{target}}(x)}{p_{\text{prop}}(x)}.$$ 

$L(\theta)$ in (8) and $L'(\theta)$ in (17) are equivalent if $p_{\text{prop}}(x)$ is the uniform distribution. Therefore, if $p_{\text{prop}}(x)$ is a uniform distribution, then our approach is equivalent to estimating the density induced by $p_{\text{target}}(x)$. Furthermore, if the shaping function is given by the exponential function as $f(\cdot) = \exp(\cdot)$, then the target distribution can be regarded as the Boltzmann distribution (LeCun et al. 2006).

5 Motion Planning by Learning the Solution Manifold in Trajectory Optimization

When applying LSMO to motion planning, we can employ various trajectory representations, including a waypoint representation as in previous studies (Zucker et al. 2013). However, for solving motion planning problems efficiently, it is effective to employ a structured trajectory representation that incorporates the desired property of a trajectory. In this section, we present a trajectory representation and an exploration strategy that makes LSMO sample-efficient in motion planning.

5.1 Trajectory Representation

Incorporating the desired property of the trajectory into the trajectory parameterization is essential to reducing the computational cost of training a neural network. For this purpose, we parameterize a trajectory using the following form:

$$\xi_\theta(q_0, q_T; w) = \xi^\text{base}(q_0, q_T) + F \xi^\text{residual}_w,$$

where $\xi^\text{base}(q_0, q_T)$ is the baseline trajectory, $\xi^\text{residual}_w$ is the term parameterized with a vector $w$, and $F$ is a time-dependent scaling matrix given by a diagonal matrix defined as

$$F = \begin{bmatrix} s(t_0) & 0 & \cdots & 0 \\ 0 & \ddots & \vdots & \vdots \\ \vdots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & s(t_T) \end{bmatrix},$$

where $s(t)$ is a time-dependent scaling function designed to satisfy $s(0) = s(1) = 0$. In our implementation, $s(t)$ is given by

$$s(t) = \begin{cases} at & \text{if } 0 \leq t < \epsilon \\ 1 & \text{if } \epsilon \leq t < 1 - \epsilon \\ 1 - at & \text{if } 1 - \epsilon \leq t \leq 1 \end{cases},$$

where $a$ and $\epsilon$ are constants. Using this parameterization, we can guarantee that the trajectories obtained from $p(\xi|x)$ satisfy the constraints of the start and goal configurations. The residual term $\xi^\text{residual}_w$ is represented by a linear combination of the basis function.

$$\xi^\text{residual}_w = \Phi w,$$

where $\Phi \in \mathbb{R}^{T \times B}$ is a feature matrix, $w \in \mathbb{R}^{B \times D}$ is the weight matrix, and $B$ is the number of basis functions. In our framework, the neural network is trained to output weight matrix $w$. The $i$th column of the feature matrix $\Phi(t)$ is given by the basis function

$$b_i^\text{log}(t) = \frac{1}{1 + \exp(\alpha(t - c_i))},$$

where $\alpha$ defines the slope of the function, and $c_i$ defines the center of the $i$th basis function.

A popular choice of the basis function for movement primitives is an exponential function given by

$$b_i^\text{exp}(t) = \exp\left(\frac{-(t - c_i)^2}{h}\right),$$

where $h$ defines the band width, and $c_i$ defines the center of the basis function. The basis functions in (23) and (24) are shown in Fig. 4. Although the form in (24) is popular, we use the basis function in (23) in our implementation because it leads to a smaller condition number of the feature matrix $\Phi$. Although our framework is not limited to a specific form of the basis function, it is important to select appropriate parameters of the basis functions to obtain a feature matrix with a low condition number. For a discussion of the condition number of feature matrix $\Phi$, please refer to the Appendix.

For convenience, we refer to the trajectory representation based on (19)-(22) as Residual Trajectory Primitives (RTPs) hereinafter. If we employ the waypoint representation, the resulting path can be jerky, particularly when a path involves a longer detour. By contrast, parameterization based on RTPs can ensure the smoothness of the trajectory as well as the constraints of the start and goal configurations. As described in the experimental section, the parameterization based on RTP can reduce the time for training the neural network for motion planning tasks.

5.2 Proposal Distribution

In our framework, it is essential to employ an appropriate proposal distribution for solving motion planning problems.
The update rule of CHOMP is given by:

$$\xi_{\text{base}} = \mathcal{N}(\xi_{\text{base}}, a\Sigma),$$

(25)

where $a$ is a constant and $\Sigma$ is the covariance matrix given by the inverse of the matrix $A^T A$ and $A$ is defined as

$$A = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & -1 & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix}.$$  

(26)

The mean of the proposal distribution $\xi_{\text{base}}$ is the baseline trajectory, and we define the baseline trajectory as the trajectory obtained by linearly interpolating the given start and goal configurations. Previous studies showed that the exploration based on this sampling strategy works well in the context of trajectory optimization (Kalakrishnan et al. 2011) and inverse reinforcement learning (Kalakrishnan et al. 2013).

5.3 Projection onto Constraint Solution Space

Although LSMO can learn the distribution of solutions, there is no guarantee that the output of $p_\theta(\xi|z)$ satisfies the desired constraint such as joint limits. For this reason, we fine-tune the output of $p_\theta(\xi|z)$ by using CHOMP (Zucker et al. 2013). The update rule of CHOMP is given by:

$$\xi^* = \arg\min_\xi \left\{ C(\xi^*) + g^T (\xi - \xi^*) + \frac{\eta}{2} \| \xi - \xi^* \|^2_M \right\},$$

(27)

where $g = \nabla C(\xi)$, $\xi^*$ is the current plan of the trajectory, $\eta$ is a regularization constant, and $\| \xi \|^2_M$ is the norm defined by a matrix $M$ as $\| \xi \|^2_M = \xi^T M \xi$.

The third term on the right-hand side of (27) can be interpreted as the trust region; it penalizes the change in the velocity profile of the motion when $M = A$ and $A$ is given by (26). By minimizing the cost function regularized with this penalty, we can obtain a trajectory that minimizes the collision cost with a minimal change in the velocity profile from the initial trajectory. In our framework, the generative model outputs trajectories with different velocity profiles for different values of the latent variable $z$. To maintain the diversity of the solutions, the change in the velocity profile before and after the fine-tuning of the trajectory must be minimized. Hence, the properties of CHOMP are suitable for our framework. Although the output of the deep generative model is fine-tuned using CHOMP in our implementation, we can also employ other existing methods such as GPMP (Mukadam et al. 2018).

5.4 Summary of the Proposed Motion Planning Algorithm

We summarize the motion planning algorithm based on LSMO in Algorithm 2. In the training phase, we sample trajectories by following the proposal distribution $\beta_{\text{pro}}(\xi)$ in (25). Subsequently, the costs of the sampled trajectories are evaluated, and the trajectories are parameterized based on the RTP. The generative model $p_\theta(w|z)$ is trained to maximize $\mathcal{L}(\theta, \psi)$ in (12). In the generation phase, the trained model $p_\theta(w|z)$ outputs the trajectory parameter $w$ for a given value of the latent variable $z$. A trajectory in configuration space $\xi$ is then recovered from $w$. If the recovered trajectory is not collision-free, then the trajectory $\xi$ is fine-tuned with CHOMP (Zucker et al. 2013). We refer to this algorithm for motion planning as Motion Planning by Learning the Solution Manifold in Trajectory Optimization (MPSM).

6 Evaluation with synthetic test functions

To evaluate the capability of LSMO to capture the solution manifold in optimization, we applied LSMO to optimization problems for synthetic test functions. In this evaluation, we trained a neural network using LSMO, and the output of the trained model was fine-tuned. To fine-tune the solutions, we employed the cross-entropy method (CEM) with a Gaussian distribution. To achieve a trust-region-based update such as CHOMP, the objective function for CEM at the $k$th iteration.
The conditions for training the neural network in LSMO are maintained to capture the diversity of solutions. The deviation from the initial estimation, which is necessary to maintain the diversity of solutions captured by LSMO. The deviation from the initial estimation is given by

\[ \eta_t = \sqrt{\eta_t^2} \]

where \( \eta_t \) is a coefficient, and \( \mu_k \) is the center of the Gaussian distribution used for sampling in the \( k \)th iteration of CEM. The second term penalizes the deviation from the initial estimation, which is necessary to maintain the diversity of solutions captured by LSMO. The conditions for training the neural network in LSMO are provided in the Appendix.

For the shaping function, we used the following form in the implementation:

\[
f(R(x)) = \begin{cases} 
\exp \left( \frac{\alpha (R(x) - R_{\text{max}})}{R_{\text{max}} - R_{\text{med}}} \right) & \text{if } R(x) \geq R_{\text{med}} \\
0 & \text{if } R(x) < R_{\text{med}} \end{cases}
\]  

(29)

where \( R_{\text{max}} \) and \( R_{\text{med}} \) are the maximum and median values, respectively, among the samples drawn from the proposal distribution. We set \( \alpha = 10 \) in this experiment.

For visualization, we used the test functions that take in two-dimensional inputs and outputs the one-dimensional values. As a baseline method, we applied CEM with a multimodal sampling distribution. In this baseline method, a mixture of 20 Gaussian distributions is used as the sampling distribution. The conditions for training the neural network are summarized in Table 6. These test functions are designed such that they have an infinite set of solutions, and the maximum and minimum values are approximately 1 and 0, respectively. Detailed definitions of the test functions are provided in the Appendix B.2.

The outputs of the model \( p_\theta(x|z) \) trained with LSMO and the solutions obtained by CEM are illustrated in Fig. 6. The outputs of \( p_\theta(x|z) \) are generated by linearly changing the

Table 1. Values of the objective function for the obtained solutions (mean ± standard deviation)

|                | Func. 1        | Func. 2        | Func. 3        | Func. 4        |
|----------------|----------------|----------------|----------------|----------------|
| LSMO w/o fine-tuning | 0.999 ± 0.021 | 0.994 ± 0.0059 | 0.889 ± 0.0897 | 0.973 ± 0.0346 |
| LSMO w/ fine-tuning  | 1.000 ± 1.2 × 10^{-5} | 1.000 ± 4.1 × 10^{-6} | 0.9991 ± 6.1 × 10^{-4} | 1.0000 ± 3.7 × 10^{-5} |
| CEM              | 1.000 ± 7.1 × 10^{-6} | 1.000 ± 2.5 × 10^{-5} | 0.9999 ± 5.0 × 10^{-5} | 0.998 ± 3.1 × 10^{-3} |
value of $z$ in $[-1.64, 1.64]$. This value of $z$ is chosen because $P(z < -1.64) = 0.05$ and $P(z < 1.64) = 0.95$ when $z \sim N(0, 1)$. In Fig. 6(a)-(h), circles represent the output of the model trained with LSMO, and the color of the circle indicates the value of the latent variable. It is evident that samples drawn from the model $p_\theta(x|z)$ correspond to the region of optimal solutions of the objective function. The output of the trained model continuously changes by continuously changing the value of the latent variable.

While CEM finds multiple solutions for objective functions, the number of solutions needs to be manually specified. Moreover, the similarity of the obtained solutions is not indicated by CEM. As shown in Figure 6(i)-(l), the solutions found by CEM that correspond to the first and second components of GMMs are separated from each other. Although there are many black-box methods for multimodal optimization, they have a common property of CEM: the similarity of solutions is not indicated, and the user would need to examine all the solutions to find the most preferable one. It is possible to find 100 solutions using CEM for test functions in these experiments, but it is be tedious for the user to check all of them. By contrast, an infinite set of solutions are modeled with a neural network, and the similarity of solutions is indicated by the value of the latent variable in LSMO. Using LSMO, the user can intuitively examine the various solutions by changing the value of the latent variable.

A limitation of LSMO indicated by this experiment is that the manifold learned by LSMO may not capture all variations of solutions, although it is clear that LSMO can capture more diverse solutions than CEM. For example, although the model trained by LSMO captures the various solutions in Fig. 6(d), there is a region of optimal points which is not covered by the outputs of the model.

The scores of the output of the model $p_\theta(x|z)$ are summarized in Table 1. The result indicates that the output of the model trained with LSMO does not necessarily correspond to an exact solution. For example, although the test function shown in Fig. 6(c) actually has a unique solution, the model trained with LSMO learns the manifold corresponding to the direction in which the gradient of the test function is gradual. As a result, the solutions before fine-tuning are not as accurate as those found by CEM. This result is natural because the output of the trained model is the result of amortized variational inference and not the optimization for each point (Kim et al. 2018; Cremer et al. 2018). Therefore, fine-tuning the output of $p_\theta(x|z)$ is necessary to obtain exact solutions. Figure 6 and Table 1 show that we can obtain accurate and diverse solutions by fine-tuning the output of $p_\theta(x|z)$. The process of fine-tuning is approximately 0.1 s for each solution, and we think that it is negligible in practice.

## 7 Evaluation of motion planning tasks

### 7.1 Implementation Details

In motion planning tasks, we used the objective function given by $R(\xi) = -C(\xi)$, where $C(\xi)$ is the cost function used in previous studies on trajectory optimization (Zucker et al. 2013) given by

$$C(\xi) = c_{\text{obs}}(\xi) + \alpha c_{\text{smoothens}}(\xi).$$

The first term in (30), $c_{\text{obs}}(\xi)$, is the penalty for collision with obstacles. Given a configuration $q$, we denote by $x_u(q) \in \mathbb{R}^3$ the position of the bodypoint $u$ in the task space. $c_{\text{obs}}(\xi)$ is then given by

$$c_{\text{obs}}(\xi) = \frac{1}{2} \sum_{t} \sum_{u \in B} c(x_u(q_t)) \left\| \frac{d}{dt} x_u(q_t) \right\|,$$

and $B$ is a set of body points that comprise the robot body. The local collision cost function $c(x_u)$ is defined as

$$c(x_u) = \begin{cases} 0, & \text{if } d(x_u) > \epsilon, \\ \frac{1}{2}(d(x_u) - \epsilon)^2, & \text{if } 0 < d(x_u) < \epsilon, \\ -d(x_u) + \frac{1}{2} \epsilon, & \text{if } d(x_u) < 0, \end{cases}$$

where $\epsilon$ is the constant that defines the margin from the obstacle, and $d(x_u)$ is the shortest distance in task space between the bodypoint $u$ and obstacles. The second term in (30), $c_{\text{smoothens}}(\xi)$, is the penalty on the acceleration defined as $c_{\text{smoothens}}(\xi) = \sum_{t=1}^{T} \| \ddot{q}_t \|^2$. To make the computation efficient, the body of the robot manipulator and obstacles are approximated by a set of spheres. We used the shaping function in (29) as in the previous experiment. The effect of the value of $\alpha$ was investigated in the following experiments.

### 7.2 Evaluation in Simulation

We first evaluated the proposed method on three tasks in a simulation with a KUKA Light Weight Robot (LWR) with...
7 DoFs. The task settings used are shown in Figures 7-9. The conditions for training the neural network in MPSM is provided in the Appendix.

To analyze the effect of the scaling parameter $\alpha$ in (29), we performed motion planning with $\alpha = 10, 20$ and $50$. In addition, to see the effect of the parameterization based on RTP, we compared the results between the parameterization based on RTP and the waypoint parameterization.

As baseline methods, we also evaluated SMTO (Osa 2020), CHOMP (Zucker et al. 2013), and STOMP (Kalakrishnan et al. 2011). SMTO was recently proposed by Osa (2020), and it finds multiple solutions for motion planning. CHOMP and STOMP find a single solution for motion planning, and we choose them as baseline methods because our motion planning adapted CHOMP and STOMP in our framework. CHOMP is used for fine-tuning the trajectory, and the exploration strategy used in STOMP is adapted for a proposal distribution in MPSM. To evaluate the computation time of MPSM, we trained the model five times with different random seeds and generated 30 samples by drawing the value of the latent variable from the uniform distribution $U([-1.28, 1.28])$. The computation time for CHOMP and STOMP was evaluated by performing the motion planning tasks with different initializations 30 times. The initial trajectories for CHOMP and STOMP were drawn from the proposal distribution $\beta_{\text{traj}}(\xi)$ in (25).

### Table 2. Number of solutions found by SMTO.

|         | Task 1 | Task 2 | Task 3 |
|---------|--------|--------|--------|
| SMTO    | 3.0 ± 0.0 | 2.0 ± 0.0 | 3.0 ± 0.0 |

7.2.1 Diversity of Solutions The results obtained using the model with the one-dimensional latent variable are presented in Figure 10. As shown, the model $p_\theta(\xi|z)$ trained with MPSM can generate various collision-free trajectories for the specified start and goal configurations. SMTO found 2 to 3 solutions as summarized in Table 2, and it is evident that MPSM found more diverse solutions. For example, the results of MPSM for Task 1 are shown in Figure 10(a). When the latent variable is one-dimensional on Task 1, the end-effector moves over the obstacle if $z = -1.28$, whereas the end-effector moves behind the obstacle if $z = 1.28$. As shown, the trajectory generated with the trained model continuously changes when the value of the latent variable $z$ is changed continuously between $z = -1.28$ and $z = 0$, and between $z = 0$ and $z = 1.28$. The same time, Figure 10(a) indicates a discontinuous change in the trajectory between $z = 0$ and $z = -0.34$. This discontinuous change is resulted from fine-tuning with CHOMP.

The distributions of the outputs generated by the model for Task 1 is shown in Figure 11 in which (a) and (b) show the distributions before and after fine-tuning, respectively. For visualization, the latent variable is uniformly obtained from $[-2, 2]$, and the dimensionality of the trajectories is reduced using t-SNE (van der Maaten and Hinton 2008). In Figure 11, the color bar indicates the value of $z$, and the black circles and triangles represent trajectories generated using $z = -1.28, 0.0$, and $1.28$, respectively. The distribution of outputs after fine-tuning with CHOMP is disconnected in some regions, as shown in Figure 11(b). This result indicates that the model $p_\theta(\xi|z)$ includes solutions from different homotopy classes.

Prior studies show that a model with a continuous latent variable can represent samples from different
reduce the computational cost. When we use the way point of the parametrization based on RTPs can significantly models exhibit comparable performance. Therefore, the use of the parameterization based on RTPs. The time required to train the training time of MPSM takes about 25 min, whereas Table 3. When using the waypoint trajectory representation, Table 3. Effect of hyperparameters on computation time in motion planning tasks.

| Task 1 | Task 2 | Task 3 |
|--------|--------|--------|
| Training | Generation | Training | Generation | Training | Generation |
| MPSM w/ RTP, 1d, $\alpha = 20$ | 5.18 ± 0.06 | 0.21 ± 0.31 | 5.26 ± 0.01 | 0.23 ± 0.31 | 5.22 ± 0.06 | 0.11 ± 0.17 |
| MPSM w/ RTP, 2d, $\alpha = 20$ | 5.21 ± 0.01 | 0.41 ± 0.55 | 5.25 ± 0.04 | 0.22 ± 0.30 | 5.22 ± 0.03 | 0.12 ± 0.17 |
| MPSM w/o RTP, 1d, $\alpha = 20$ | 24.4 ± 0.16 | 0.68 ± 0.73 | 24.6 ± 0.12 | 0.24 ± 0.04 | 24.5 ± 0.03 | 0.24 ± 0.04 |
| MPSM w/ RTP, 1d, $\alpha = 10$ | 5.24 ± 0.07 | 0.42 ± 0.43 | 5.21 ± 0.04 | 0.19 ± 0.26 | 5.10 ± 0.03 | 0.50 ± 0.31 |
| MPSM w/ RTP, 1d, $\alpha = 50$ | 5.21 ± 0.06 | 0.33 ± 0.53 | 5.17 ± 0.07 | 0.36 ± 0.37 | 5.08 ± 0.04 | 0.06 ± 0.01 |
| SMTO | - | 51.3 ± 0.69 | - | 50.5 ± 1.0 | - | 53.4 ± 1.7 |
| CHOMP | - | 1.8 ± 1.9 | - | 0.52 ± 0.22 | - | 7.3 ± 11.4 |
| STOMP | - | 7.91 ± 24.3 | - | 1.24 ± 2.23 | - | 44.1 ± 53.0 |

Table 4. Effect of hyperparameters on scores of the trajectories before fine-tuning. Higher is better.

| Task 1 | Task 2 | Task 3 |
|--------|--------|--------|
| MPSM w/ RTP, 1d, $\alpha = 20$ | -2.80 ± 2.16 | -2.61 ± 2.78 | -2.22 ± 0.18 |
| MPSM w/ RTP, 2d, $\alpha = 20$ | -3.83 ± 3.23 | -2.60 ± 2.83 | -2.28 ± 0.19 |
| MPSM w/o RTP, 1d, $\alpha = 20$ | -2.69 ± 1.78 | -2.04 ± 2.52 | -2.27 ± 0.14 |
| MPSM w/ RTP, 1d, $\alpha = 10$ | -3.74 ± 2.53 | -2.40 ± 2.52 | -2.70 ± 0.49 |
| MPSM w/ RTP, 1d, $\alpha = 50$ | -3.04 ± 2.94 | -3.86 ± 3.37 | -1.79 ± 0.16 |

classes/clusters. For example, Kingma and Welling (2014) reported that a VAE with a continuous latent variable can model different hand-written digit images in the MNIST dataset, which contains multiple distinctive classes. When samples from different classes are modeled using a VAE with a continuous latent variable, the datapoints are often interpolated, e.g., the model can generate an image between “2” and “9” when trained with the MNIST dataset. Likewise, the proposed method modeled trajectories from different homotopy classes, as shown in the experiments, and the trajectories were interpolated in the learned latent space. Consequently, the distribution of the outputs is continuously connected, as shown in Figure 11(a). However, in motion-planning problems, the interpolation of solutions from different homotopy classes can result in non-collision-free trajectories. Therefore, we project the non-collision-free trajectories onto the collision-free solution space by performing fine-tuning using CHOMP. After fine-tuning, the distribution of the obtained solutions was disconnected, as shown in Figure 11(b).

The task-space trajectories of the end-effector, which are shown in Figure 12, imply that two clusters of solutions are available for Task 1. These results indicate that the model trained with LSMO represents solutions from different homotopy classes. Detailed results of Task 2 and 3 are provided in Appendix C.1; they support the discussion presented in this section.

7.2.2 Computation Time and Effect of Hyperparameters
The computation time for motion planning is summarized in Table 3. When using the waypoint trajectory representation, the training time of MPSM takes about 25 min, whereas the training time of MPSM takes 5.5 min when using the parameterization based on RTPs. The time required to generate a solution shown in Table 3 indicates that these models exhibit comparable performance. Therefore, the use of the parametrization based on RTPs can significantly reduce the computational cost. When we use the way point parameterization, the value of each element may exhibit a large variance. However, the variance can be reduced using RTPs because only the residual from the baseline trajectory is to be learned in RTPs.

The results in Table 3 also indicate that $\alpha = 20$ outperforms $\alpha = 10$ and 50 in the sense that the time required for fine-tuning was the minimum across the tasks. The scores of the trajectories generated from the model shown
in Table 4 also indicate that $\alpha = 20$ outperforms $\alpha = 10$ and 50. When the scaling factor $\alpha$ is larger, the relative importance of samples with higher scores becomes larger, which encourages the estimated density to be focused on the samples with high scores. However, when the difference in the importance weight among samples is larger, the variance of estimating the loss function for training the neural network becomes larger, which may lead to unstable training. Therefore, it is necessary to select an appropriate value for the scaling factor $\alpha$.

7.3 Experiments with Real Robot

7.3.1 Motion Planning for a Real Robot To verify that the proposed algorithm is applicable to real robots, we performed motion-planning experiments using a real robot. We used Cobotta (Denso Wave Inc.), which has six DoFs, in the experiment. The task settings are shown in Figure 13; in this study, we refer to the tasks as Tasks 4, 5, and 6, respectively.

The results with a one-dimensional latent variable are shown in Figure 14. For Task 4, the end-effector moves behind the obstacle when $z = -1.28$, whereas the end-effector moves over the obstacle when $z = 1.28$, as shown in Figure 14(a). Figure 15 shows the solutions obtained...
for Task 4 in the task space; as shown, the collision-free trajectories generated by the trained neural network changed continuously as the value of the latent variable was changed. This result indicates that the neural network trained with MPSM captured a set of homotopic collision-free trajectories. As shown in Figure 14(b) and (c), MPSM also obtained diverse solutions for Tasks 5 and 6. When we applied SMTO to these tasks, two or three solutions were obtained. Therefore, it is evident that MPSM can obtain more diverse solutions than SMTO. Please refer to Appendix C.3 for the solutions obtained by SMTO.

To demonstrate the effect of the dimensionality of the latent variable, we present the solutions obtained for Task 4 using the model with the two-dimensional latent variable in Figure 16. The distributions of solutions obtained by MPSM for Task 4 were visualized using t-SNE, as shown in Figure 17. The results suggest that the model with the two-dimensional latent variable represents more diverse solutions than the model with the one-dimensional latent variable. Regarding the fine-tuning of the output of the neural network, it was not necessary to fine-tune the outputs when the latent variable was one-dimensional for all tasks. Meanwhile, when the latent variable was two-dimensional, fine-tuning was occasionally necessary. This result indicates that there is a trade-off between the diversity and accuracy of solutions when selecting the dimensionality of the latent variable. It can also be seen that the difference in the information encoded in the two channels are not clear when the latent variable is two-dimensional. For example, Figure 16 implies that both $z_0$ and $z_1$ encode the variation in the height of the end-effector. Therefore, the results imply that increasing the dimensionality of the latent variable may complicate the process for the user to examine solutions captured by MPSM, although this may lead to a greater diversity of solutions. The results of Tasks 5 and 6 are provided in Appendix C.3; they also support the discussions presented in this section.

### 7.3.2 Adaptation to Scene Change

As discussed by Kumar et al. (2020) in the context of reinforcement learning, obtaining diverse solutions can lead to robustness against scene changes. In our framework, diverse solutions can be obtained for a motion planning problem. When a few obstacles are added to the original scene, feasible solutions can be obtained in a set of the solutions for the original scenario, although a subset of solutions are disabled. In the scenario shown in Figure 18, obstacles were added in the scenario of Task 6, which is shown in Figure 13(e) and (f). Our system obtained solutions shown in Figure 18 using the model trained for Task 6. In our implementation, we can verify the collision of a trajectory in approximately 36 ms and quickly identify usable trajectories from those obtained for the original scenario. Even if the trajectory identified for the original scenario is not directly usable for a new scenario, it can be serve as a descent initial trajectory for trajectory optimization when the change in the scenario is insignificant. Therefore, the model need not to be re-trained from scratch if the change in the scenario is insignificant.

### 8 Discussion

The motivation of this work is to allow the user to select the preferable solutions because the objective function used in motion planning may not properly reflect the user preference. If the end-effector moves over the obstacle as in the left-most frame of Figure 10(a), the user may find it scary. However, programming such a preference is not trivial, and dealing with all such preferences is actually challenging. Therefore, we provide diverse solutions for the user and let the user...
select one, rather than programming the user preferences. We believe that our framework enhances the usability of a motion planner and helps make robots usable in daily life. Although we focused on motion planning problems in this study, the concept of learning diverse solutions by learning latent representations is applicable to other domains. For example, in our recent study, we developed a reinforcement learning method that obtains diverse solutions by learning latent representations; subsequently we applied it to continuous control tasks (Osa et al. 2021).

The tasks employed in this study involve large free space because our framework is beneficial in such circumstances. Under the existence of a large free space, there are diverse ways to avoid obstacles and reach the desired position. In such cases, the user will need to examine different solutions to perform the motion and select the type of motion. In our framework, the variation of the trajectories is encoded in a low-dimensional latent variable, and this allows the user to intuitively examine different solutions. If the free space is limited, then a possible variation of the solutions is limited. In such cases, the user does not need to examine diverse solutions, and she/he is encouraged to employ sophisticated existing motion planning frameworks to find a single solution efficiently and robustly, such as TrajOpt (Schulman et al. 2014), GPMP (Mukadam et al. 2018) and BIT* (Gammell et al. 2020).

Our method trains a generative model for solutions in motion planning. Various methods for training deep generative models have been developed in recent studies (Goodfellow et al. 2014; Chen et al. 2016; Arjovsky et al. 2017; Dupont 2018); prior work such as (Bengio et al. 2013; Berthelot et al. 2018; Verma et al. 2019) investigated how to obtain meaningful latent representations in the context of unsupervised learning. Although we investigated the manner in which techniques for deep generative models can be leveraged for motion planning, methods to obtain meaningful latent representations for motion-planning problems must be further investigated. A limitation of our framework is that it takes approximately 5 min to train a neural network, which is not required for methods that do not use a neural network. As obtaining an infinite set of diverse solutions is challenging, it is natural that there is a trade-off between the diversity of solutions and the computation time. When using existing motion planning methods, it is necessary to manually tune the objective function or explore different random seeds in order to obtain different types of solutions. Compared with such efforts, we think that the time required for training a neural network is negligible in practice.

A possible extension of this work is to learn both continuous and discrete latent variables to explicitly learn multiple separate sets of solutions. This extension should be possible with the Gumbel-Softmax trick (Jang et al. 2017; Maddison et al. 2017). Regarding the objective function, we did not explicitly incorporate topology-based representations in the objective function, although the trained model captured diverse solutions from homotopic classes. Ivan (2013) investigated the topology-based representations to describe the geometric relations between robots and objects. Incorporating such representations with LSMO will be an interesting research topic. Another possible extension is to make the neural network conditioned on scene features in such a way that the neural network can generate a trajectory for unseen situations. This is an important research direction to remove the necessity of training a neural work for each situation. However, training a neural network to generate a collision-free trajectory for unseen situations is challenging (Srinivas et al. 2018), and training a neural network for generating diverse collision-free trajectories for unseen situations is even more challenging. We will investigate this extension in future work.

9 Conclusion

In this study, we presented LSMO, which is an algorithm for learning an infinite set of solutions in optimization. In our framework, diverse solutions are captured by learning latent representations of solutions. We derived the proposed algorithm by considering the variational lower bound of the expected score of solutions. We then adapted LSMO to motion planning problems and developed a novel motion planning algorithm, which we referred to as MPSM. Our approach can be interpreted as training a deep generative model of collision-free trajectories for motion planning. In our experiments, we show that a set of homotopic solutions can be obtained with MPSM on motion planning tasks, which involve hundreds of parameters. Our framework for learning the solution manifold gives users an intuitive way to go through various solutions by changing the values of the latent variables. We believe that the approach of learning the solution manifold in optimization can enhance the usability of motion planners in robotics by providing diverse solutions for the user. In future work, we will investigate how to incorporate other deep generative models such as GANs in our framework.

Funding

This work was partially supported by JSPS KAKENHI Grant Number 19K20370.

References

Agrawal S, Shen S and v d Panne M (2014) Diverse motions and character shapes for simulated skills. *IEEE Transactions on Visualization and Computer Graphics* 20(10): 1345–1355.

Amari S (2016) *Information geometry and its applications*. Springer.

Argall BD, Chernova S, Veloso M and Browning B (2009) A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57(5): 469–483.
A Condition Number of Feature Matrix

To make the paper self-contained, we describe how the condition number of $\Phi$ indicates the numerical stability of modeling trajectories with a neural network. The condition number of a matrix $A$ is defined as

$$\kappa(A) = \|A\| \cdot \|A^{-1}\|$$

(33)

for which the value depends on the norm we use. In the following discussion, we assume that we use the $\ell_2$-norm.

We consider the problem of estimating $w$ and compute $\xi = \Phi w$. This computation appears in trajectory generation with DMPs and ProMPs. When the error of estimating $w$ is
given by $\Delta w$, the relative error of estimating $w$, which is the ratio of estimation error $\Delta w$ and the true value of $w$, is given by

$$r_w = \frac{\|\Delta w\|_2}{\|w\|_2}.$$ 

Likewise, we denote by $\Delta \xi$ the error of estimating $\xi$, and the relative error of estimating $\xi$ is given by

$$r_\xi = \frac{\|\Delta \xi\|_2}{\|\xi\|_2}.$$ 

The ratio of $r_w$ and $r_\xi$ indicates how the error of estimating $w$ is propagated to the error of estimating $\xi$. We can obtain the following relation using $\xi = \Phi w$:

$$\frac{r_\xi}{r_w} = \frac{\|\Delta \xi\|_2}{\|\xi\|_2} \frac{\|\Delta w\|_2}{\|w\|_2} = \frac{\|\Phi \Delta w\|_2}{\|\Phi w\|_2} \frac{\|w\|_2}{\|w\|_2} \leq \max \frac{\|\Phi \Delta w\|_2}{\|\Delta w\|_2} \frac{\|\Phi^{-1} w\|_2}{\|w\|_2} = \|\Phi\|_2 \cdot \|\Phi^{-1}\|_2 = \kappa(\Phi).$$

Therefore, condition number $\kappa(\Phi)$ is the upper bound of $\frac{r_\xi}{r_w}$.

Table 5 shows a comparison of the condition number $\kappa(\Phi)$ between the two basis functions. When using the basis function $b_{tanh}(s)$ for $\Phi$, the condition number is $\kappa(\Phi) \approx 10^{17}$ for $h = 0.1$. This large condition number indicates that an error in $w$ can be significantly magnified when estimating $\xi$. For example, suppose that the order of the value is given as $\|\xi\|_2 \approx 10$ and $\|w\|_2 \approx 10$. If the necessary precision for estimating $\xi$ is $\|\Delta \xi\|_2 \approx 0.1$, then the order of the estimation error of $w$ should be $\|\Delta w\|_2 \approx 10^{-18}$, which is problematic when we train a neural network to estimate $w$ for planning $\xi$. The large condition number of $\Phi$ may not be problematic when the solution can be obtained using a closed form as in DMPs and ProMPs because we can obtain a solution with high accuracy in a single matrix calculation. However, when we train a neural network that estimates $w$, minimizing the loss function with a stochastic gradient descent will require numerous iterations to achieve such high precision.

When using the basis function in (23) with $\alpha = 50$ for the feature matrix, the condition number is $\kappa(\phi) \approx 1000$. In the above example, the necessary precision is $\|\Delta w\|_2 \approx 10^{-4}$, which is achievable in the training of a neural network. We employed the basis function in (23) in our experiments with RTP, although other forms of the basis function can be used as long as the condition number is sufficiently small.

It is worth noting that scaling the feature matrix $\Phi$ does not change the condition number as $\kappa(\Phi) = \kappa(a\Phi)$, where $a$ is an arbitrary real number. Therefore, the numerical instability caused by the large condition number $\kappa(\Phi)$ cannot be resolved by scaling the feature matrix $\Phi$.

### B Experiment Details

#### B.1 Conditions for training a neural network for tasks with synthetic test functions

We provide the network architecture and training parameters for the tasks with synthetic test functions in Table 6. The implementation of VAE is based on the implementation provided by Dupont (2018).

#### B.2 Definitions of test functions

The definitions of the test functions used in the experiment are as follows. The figures plot the range $x_1 \in [0, 2]$ and $x_2 \in [0, 2]$.

The test function 1 is given by

$$R(x_1, x_2) = \exp(-2d)$$

where

$$d = \begin{cases} \frac{(x_2 - 1.05)^2 + (x_1 - 0.5)^2}{(0.04 + 0.8)}, \quad \text{if } x_1 < 0.5, \\ \frac{(x_2 - 0.75)^2 + (x_1 - 1.5)^2}{(0.04 + 0.8)}, \quad \text{if } x_1 > 1.5. \end{cases}$$

The toy function 2 is given by

$$R(x_1, x_2) = \exp(-2d)$$

where

$$d = \frac{|x_2 - 1.5|^2 + (x_1 + 1)^2 - 2.5|.$$}

The toy function 3 is given by

$$R(x_1, x_2) = \exp\left(-2(d + 0.2x_2 + 0.14)\right)$$

where

$$d = \begin{cases} \frac{((x_2 - 0.94)^2 + (x_1 + 0.7)^2)^{0.5}}{(0.2x_1 - x_2 + 0.8)}, \quad \text{if } x_1 < 0.7, \\ \frac{(x_2 - 1.08)^2 + (x_1 - 1.4)^2}{(0.04 + 0.8)}, \quad \text{if } x_1 > 1.4. \end{cases}$$

The toy function 4 is given by

$$R(x_1, x_2) = \exp(-2d)$$

where

$$d = |(x_2 - 1)^2 + (x_1 - 1)^2 - 0.5|.$$}

#### B.3 Conditions for training a neural network for motion planning tasks

We provide the network architecture and training parameters for the motion planning tasks in Table 6. The information capacity $C_z$ in (13) is linearly increased from 0 to 5 during the training, using the implementation of joint VAE provided by Dupont (2018).

### C Additional results on motion planning tasks

#### C.1 Solutions Found for Tasks 2 and 3

Additional results on simulated environments are shown in Figure 19–33. The results of Tasks 2 and 3 support the discussion in the main manuscript. For example, MPSM learned trajectories from different homotopic classes on Task 2. As shown in Figure 24(a), the end-effector moves over the table if $z = -1.28$, whereas the end-effector moves behind the obstacle if $z = 1.28$. The trajectory changes continuously as the value of $z$ changes, but a discontinuous change in the trajectory occurs between $z = 0.34$ and $z =$
0.72, as shown in Fig. 24(a). Actually, the distribution of solutions is disconnected after fine-tuning as shown in Fig. 26. Figure 28 also implies that there are two clusters of solutions for Task 2. Meanwhile, it was not necessary to fine-tune the output of \( p_\theta(z|\xi) \) for Task 3, and the distribution of the solutions was continuously connected, as shown in Fig. 31.

### C.2 Effect of Dimensionality of Latent Variable

Regarding the dimensionality of the latent variable, increasing the dimensionality of the latent variable did not necessarily lead to an increase in the diversity of solutions. Although the use of a two-dimensional latent variable led to more diverse solutions than the one-dimensional latent variable for Task 2, this was not the case for Tasks 1 and 3. For Task 2, the trajectories shown in Figure 24(b) indicate that the two channels \( z_0 \) and \( z_1 \) encode different trajectories. When the value of \( z_0 \) is varied, the posture of the manipulator changes while maintaining the height of the end-effector during the motion. In contrast, the height of the end-effector during the motion changes when the value of \( z_1 \) is varied. When examining the trajectories found for Task 1, which are visualized in Figure 19(b), the effect of changing the value of \( z_0 \) is not clear. In Figure 20, we show the KL divergence \( D_{KL}(q_\psi(z|x)||p(z)) \), which is the upper bound of the mutual information between \( z \) and \( x \) (Dupont 2018). The plot of the KL divergence also indicates that channel \( z_1 \) encodes more information than channel \( z_0 \) on Task 1. For Task 3, although two channels encode variations of trajectories, as shown in Figure 29(b), the difference of variations encoded in \( z_0 \) and \( z_1 \) is not clear. Figure 30 indicates that comparable amount of information is encoded in \( z_0 \) and \( z_1 \). These observations suggest that the information encoded in \( z_0 \) and \( z_1 \) is entangled for Task 3. Therefore, the one-dimensional latent variable was sufficient to model the diversity of solutions for these tasks.

### C.3 Additional Results with Real Robot Experiments

Solutions found for Tasks 4, 5 and 6 are shown in Figure 34–42. For Task 5, the height of the end-effector continuously changes as the value of the latent variable changes, as shown in Figure 37. For Task 6, when the latent variable is one-dimensional, the end-effector avoids the shelf from the right-hand side if \( z = -1.28 \), whereas the end-effector avoids the shelf from the left-hand side if \( z = 1.28 \), as shown in Figure 40. Solutions found by SMTO for Tasks 4, 5, and 6 are shown in Figures 35, 38, and 41, respectively. It is evident that MPSM found more diverse solutions than SMTO.
Figure 19. Solutions generated from $p_{\theta}(\xi \mid z)$ with different values of latent variable $z$ on Task 1.

Figure 20. KL divergence during training on Task 1.

Figure 21. Visualization of distribution of solutions on Task 1. Dimensionality is reduced using t-SNE. Color bar indicates value of $z$.

Figure 22. Three solutions found by SMTO for Task 1.

Figure 23. Trajectories in task space for Task 1. Result with one-dimensional latent variable. 20 trajectories are generated by linearly interpolating between $z = -1.28$ and $z = 1.28$. 

(a) Result with one-dimensional latent variable.

(b) Result with two-dimensional latent variable. $z_1 = 0$ in top row, and $z_0 = 0$ in bottom row.
(a) Result with one-dimensional latent variable.

(b) Result with two-dimensional latent variable. \( z_1 = 0 \) in top row, and \( z_0 = 0 \) in bottom row.

**Figure 24.** Solutions generated from \( p_\theta(\xi | z) \) with different values of latent variable \( z \) on Task 2.

(a) Results with one-dimensional latent variable.
(b) Results with two-dimensional latent variable.

**Figure 25.** KL divergence during training on Task 2.

(a) Distribution of solutions before fine-tuning.
(b) Distribution of solutions after fine-tuning.

**Figure 26.** Visualization of distribution of solutions on Task 2. Dimensionality is reduced using t-SNE. Color bar indicates value of \( z \).

**Figure 27.** Two solutions found by SMTO for Task 2.

**Figure 28.** Trajectories in task space for Task 2. Result with one-dimensional latent variable. 20 trajectories are generated by linearly interpolating between \( z = -1.28 \) and \( z = 1.28 \).
Osa

Figure 29. Solutions generated from \( p_\theta(\xi | z) \) with different values of latent variable \( z \) on Task 3.

Figure 30. KL divergence during the training on Task 3.

Figure 31. Visualization of distribution of solutions on Task 3. Dimensionality is reduced using t-SNE. The color bar indicates the value of \( z \). As fine-tuning was not necessary, distribution of solution did not change between before and after the fine-tuning.

Figure 32. Three solutions found by SMTO for Task 3.

Figure 33. Trajectories in task space for Task 3. Result with one-dimensional latent variable. 20 trajectories are generated by linearly interpolating between \( z = -1.28 \) and \( z = 1.28 \).
Figure 34. Solutions generated from $p_\theta(\xi | z)$ with different values of latent variable $z$ on Task 4.

Figure 35. Three solutions generated by SMTO for Task 4.

Figure 36. Distribution of solutions on Task 4. Dimensionality is reduced using same transformation in (a) and (b). The color bar indicates value of $z$ in (a) and $z_0$ in (b), respectively.
Figure 37. Solutions generated from $p_\theta(\xi|z)$ with different values of latent variable $z$ on Task 5.

Figure 38. Three solutions generated by SMTO for Task 5.

Figure 39. Distribution of solutions on Task 5. Dimensionality is reduced using same transformation in (a) and (b). Color bar indicates value of $z$ in (a) and $z_0$ in (b), respectively.
(a) Result with one-dimensional latent variable.

(b) Trajectories in task space. Results with the one-dimensional latent variable.

(c) Result with two-dimensional latent variable.

Figure 40. Solutions generated from $p_{\theta}(\xi|z)$ with different values of latent variable $z$ on Task 6.

Figure 41. Two solutions generated by SMTO for Task 6.

Figure 42. Distribution of solutions on Task 6. Dimensionality is reduced using same transformation in (a) and (b). Color bar indicates value of $z$ in (a) and $z_0$ in (b), respectively.