Goal-Conditioned Variational Autoencoder Trajectory Primitives with Continuous and Discrete Latent Codes

doi: https://doi.org/10.1007/s42979-020-00324-7

SN Computer Science manuscript No.
(will be inserted by the editor)

Goal-Conditioned Variational Autoencoder Trajectory Primitives with Continuous and Discrete Latent Codes

Takayuki Osa* · **Shuehi Ikemoto**

Received: date / Accepted: date

Abstract Imitation learning is an intuitive approach for teaching motion to robotic systems. Although previous studies have proposed various methods to model demonstrated movement primitives, one of the limitations of existing methods is that the shape of the trajectories are encoded in high dimensional space. The high dimensionality of the trajectory representation can be a bottleneck in the subsequent process such as planning a sequence of primitive motions. We address this problem by learning the latent space of the robot trajectory. If the latent variable of the trajectories can be learned, it can be used to tune the trajectory in an intuitive manner even when the user is an expert. We propose a framework for modeling demonstrated trajectories with a neural network that learns the low-dimensional latent space. Our neural network structure is built on the variational autoencoder (VAE) with discrete and continuous latent variables. We extend the structure of the existing VAE to obtain the decoder that is conditioned on the goal position of the trajectory for generalization to different goal positions. Although the inference performed by VAE is not accurate, the positioning error at the generalized goal position can be reduced to less than 1 mm by incorporating the projection onto the solution space. To cope with requirement of the massive training data, we use a trajectory augmentation technique inspired by the data augmentation commonly used in the computer vision community. In the proposed framework,

This work is supported by JSPS KAKENHI Grant Number 19K20370 and 18H01410.

T. Osa
Department of Human Intelligence Systems & Neuromorphic AI Hardware Research Center, Kyushu Institute of Technology, Fukuoka, Japan
RIKEN center for Advanced Intelligence Project, Tokyo, Japan
E-mail: osa@brain.kyutech.ac.jp

S. Ikemoto
Department of Human Intelligence Systems & Neuromorphic AI Hardware Research Center, Kyushu Institute of Technology, Fukuoka, Japan
E-mail: ikemoto@brain.kyutech.ac.jp

the latent variables that encodes the multiple types of trajectories are learned in an unsupervised manner, although existing methods usually require label information to model diverse behaviors. The learned decoder can be used as a motion planner in which the user can specify the goal position and the trajectory types by setting the latent variables. The experimental results show that our neural network can be trained using a limited number of demonstrated trajectories and that the interpretable latent representations can be learned.

Keywords Learning from demonstrations · Variational autoencoder

1 INTRODUCTION

Imitation learning (IL) [1] is an approach that can achieve such intuitive motion teaching. In IL, a user demonstrates how a task is performed, e.g. through kinesthetic teaching (Fig. 1), and the system learns how to generalize demonstrated trajectories to different conditions. Previous studies have proposed various ways to model demonstrated trajectories such as Dynamic Movement Primitive (DMP) [2] and Probabilistic Movement Primitive (ProMP) [4] and Kernelized Movement Primitive (KMP) [5]. Although these methods can cope with complex generalization of the demonstrated trajectories, one of the limitations of existing methods is that the shape of the trajectories are encoded in high dimensional space. A trajectory of a robotic manipulator is often high-dimensional. For example, if we represent a trajectory of a manipulator with 7 DoFs as 100 way points, a trajectory will be represented as a vector with 700 dimensions. Even if we project a trajectory onto a weight space as in existing methods such as DMP [2], the trajectory will be encoded as a weight vector with hundreds of dimensions. The high dimensionality of the trajectory representation can be a bottleneck in the subsequent process such as planning a sequence of primitive motions. We address this problem by learning the latent space of the robot trajectory. If the low-dimensional latent variable of the trajectories can be learned, it can be used to tune the trajectory when planning and optimizing a sequence of primitive trajectories. In addition, by encoding the trajectory type into the latent variable, we can model multiple trajectory types with a single model, although existing frameworks usually require separate multiple models to represent multiple behaviors.

Deep learning, which has contributed to recent advances in machine learning, is a powerful tool for coping with high-dimensional data. Moreover, it can be used to learn latent representations that capture meaningful information as low-dimensional data [6, 7]. However, neural networks are notorious in that they require thousands of samples for training. In the context of imitation learning, it is costly to collect data of human demonstrations. Therefore, the number of the available expert trajectories is often limited. To leverage the power of neural networks in imitation learning, it is essential to address the requirement of massive training data.

In this study, we present a framework for modeling multiple types of demonstrated trajectories with a single neural network by learning latent represen-

Fig. 1: Modeling the demonstrated trajectory through kinesthetic teaching is essential in imitation learning.

tations. The proposed framework does not require information on the motion type of the demonstrated trajectories, and the model is trained in an unsupervised manner. To address the issue of the size of the training data, we propose a trajectory augmentation trick that is inspired by the data augmentation commonly used in the computer vision community. We also extend the structure of the joint VAE proposed in [8] to obtain a decoder that is conditioned on the goal position of the trajectory for generalization to different goal positions. In this framework, different types of trajectories are encoded in the discrete latent variable and the continuous variable interpolates the input trajectories. The learned decoder can be used as a motion planner in which the user can specify the goal position and the trajectory types by setting the latent variables. The experimental results show that the proposed neural network can be trained with a limited number of demonstrated trajectories and learns interpretable low-dimensional representations. We think that our framework will lead to the development of an imitation-learning-based robotic system in which a user can select and modify the shape of a planned trajectory by changing low-dimensional latent variables.

2 RELATED WORK

Imitation learning is a class of methods for learning the behavior demonstrated by experts [1], and it is considered an intuitive approach that facilitates the teaching of desired motions to a robotic system. Motion planning with imitation learning has been exploited in practical applications such as robotic surgery [13]. Previous studies on movement primitives have proposed various frameworks for modeling demonstrated trajectories, e.g. DMP [2], ProMP [4], and KMP [5]. These methods enable the generalization of trajectories to different goal positions while maintaining the topological features of the demonstrated trajectories. However, these methods do not learn lower-dimensional latent representations of demonstrated trajectories, which would enable intuitive user-system interaction. Studies on deep generative models [6–8] show

that it is possible to learn intuitive latent representations of data. The original study of VAE shows that VAE can learn the latent dimension which corresponds to the facial expression and that the face image can be continuously changed from a "sad" face to an "angry" face by changing the value of latent variable [6]. In the context of imitation learning, it will be possible to generate a new "middle" behavior by learning latent representations of demonstrated trajectories.

The latent space of robotic trajectories has been leveraged in the context of imitation learning. Studies such as [14, 15] employed Gaussian Process Latent Variable Model [16] and Principal Component Analysis to learn a low-dimensional latent space to achieve efficient learning. However, it is not trivial to learn both continuous and discrete latent variables using these methods. In contrast, the autoencoder used in this study learns both discrete and continuous latent variables.

Recent advances in reinforcement learning (RL) has made it possible to apply deep learning methods to robotic manipulation. The study in [17] trained a neural network to generate a motor torque command from image inputs. Studies such as [18] proposed algorithms to learn neural network policies using deep RL for real robots. In these recent studies, the use of neural networks is often focused on Markov Decision Process (MDP) settings, where the agent performs learning to generate a control input from a given state under stochastic dynamics. The study in [19] is related to our work in that a neural network learns to imitate demonstrated behaviors through learning the latent variable for continuous control tasks. However, the use of neural networks for planning problems is still unexplored. In planning problems, it is usually unnecessary to consider a stochastic state transition as in the standard RL setting [20]. Instead, a desired trajectory is planned for a given deterministic state transition, which is actually valid assumption in practice. Many industrial robots have solid position/velocity controllers, and there is no need to learn low-level control. It is natural to consider that the system is fully actuated in these cases. Therefore, we think that control of industrial robots requires an algorithm that addresses the planning problem rather than the continuous control problem.

A few recent studies have addressed the application of neural networks to manipulation planning. The study in [21] proposed a neural network architecture that predicts the deformation of objects after a simple manipulation. To train the neural network, hundreds of manipulation trajectories were recorded. The necessity of massive training data has been an issue of applying deep learning to motion planning. In our study, we propose a technique for generating synthetic trajectories to address this issue.

3 AUTOENCODER TRAJECTORY PRIMITIVES

We present the proposed autoencoder to model the demonstrated trajectories, which we refers to as a *autoencoder trajectory primitive* (ATP). The network

architecture and the objective function are described. Then the process of creating a dataset with a sufficient number of trajectories from a limited number of demonstrated trajectories is described.

3.1 Objective Function and Architecture

We denote by $\boldsymbol{q} \in \mathbb{R}^d$ a configuration of a robot manipulator, where d is a number of joints. A trajectory given by a sequence of configurations is denoted by $\boldsymbol{\xi} = [\boldsymbol{q}_0, \dots, \boldsymbol{q}_T]$ where T is the number of time steps. We assume that a dataset of trajectories $\mathcal{D} = \{\boldsymbol{\xi}_i\}_{i=1}^N$ is given. It is also assumed that N is a sufficiently large to train a neural network. We describe the process of constructing such a dataset from a limited number of demonstrated trajectories in the next section. In this study, we aim to learn the latent space of a given dataset. The continuous and discrete latent variables are denoted by $\boldsymbol{z}$ and $\boldsymbol{c}$, respectively. We consider the variational autoencoder framework in which the the posterior/encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}, \boldsymbol{c}|\boldsymbol{\xi})$ is represented by a neural network parameterized by a vector $\boldsymbol{\phi}$, and the likelihood/decoder $p_{\boldsymbol{\theta}}(\boldsymbol{\xi}|\boldsymbol{z}, \boldsymbol{c})$ is represented by a neural network parameterized by a vector $\boldsymbol{\theta}$. For learning latent variables, β-VAE model [22] is often employed in previous studies. When learning the continuous and discrete variable $\boldsymbol{z}$ and $\boldsymbol{c}$, the objective function of the β-VAE model [22] is given by

$$\mathcal{J}_\beta = \mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{D}} \left[\mathcal{L}_\beta(\boldsymbol{\theta}, \boldsymbol{\phi}) \right], \tag{1}$$

where $\mathcal{L}_\beta(\boldsymbol{\theta}, \boldsymbol{\phi})$ is given by

$$\begin{aligned} \mathcal{L}_\beta(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}, \boldsymbol{c}|\boldsymbol{\xi})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{\xi}|\boldsymbol{z}, \boldsymbol{c})] \\ -\beta D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z}, \boldsymbol{c}|\boldsymbol{\xi})||p(\boldsymbol{z}, \boldsymbol{c})), \end{aligned} \tag{2}$$

where $q_{\boldsymbol{\phi}}(\boldsymbol{z}, \boldsymbol{c}|\boldsymbol{\xi})$ is the joint posterior, $p(\boldsymbol{z}, \boldsymbol{c})$ is the prior, $p_{\boldsymbol{\theta}}(\boldsymbol{\xi}|\boldsymbol{z}, \boldsymbol{c})$ is the likelihood, and $D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z}, \boldsymbol{c}|\boldsymbol{\xi})||p(\boldsymbol{z}, \boldsymbol{c}))$ is the KL divergence between $q_{\boldsymbol{\phi}}(\boldsymbol{z}, \boldsymbol{c}|\boldsymbol{\xi})$ and $p(\boldsymbol{z}, \boldsymbol{c})$. However, the latent variable learned by β-VAE is often non-intuitive. For learning disentangled continuous and discrete representations, Dupont[8] proposed the objective given as

$$\mathcal{J}_{\mathrm{joint}} = \mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{D}} \left[\mathcal{L}_{\mathrm{joint}}(\boldsymbol{\theta}, \boldsymbol{\phi}) \right], \tag{3}$$

where $\mathcal{L}_{\mathrm{joint}}(\boldsymbol{\theta}, \boldsymbol{\phi})$ is given by

$$\begin{aligned} \mathcal{L}_{\mathrm{joint}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}, \boldsymbol{c}|\boldsymbol{\xi})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{\xi}|\boldsymbol{z}, \boldsymbol{c})] \\ -\gamma \left| D_{\mathrm{KL}}\big(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{\xi})||p(\boldsymbol{z})\big) - C_{\boldsymbol{z}} \right| \\ -\gamma \left| D_{\mathrm{KL}}\big(q_{\boldsymbol{\phi}}(\boldsymbol{c}|\boldsymbol{\xi})||p(\boldsymbol{c})\big) - C_{\boldsymbol{c}} \right|, \end{aligned} \tag{4}$$

where $C_{\boldsymbol{z}}$ and $C_{\boldsymbol{c}}$ represent the information capacity, and γ is a coefficient. In our framework, we can compute the end-effector position at the goal configuration $\boldsymbol{x}_g$. This supervised information $\boldsymbol{x}_g$ is incorporated into the proposed neural network to obtain the decoder $p_{\boldsymbol{\theta}}(\boldsymbol{\xi}|\boldsymbol{z}, \boldsymbol{c}, \boldsymbol{x}_g)$ conditioned on $\boldsymbol{x}_g$. This

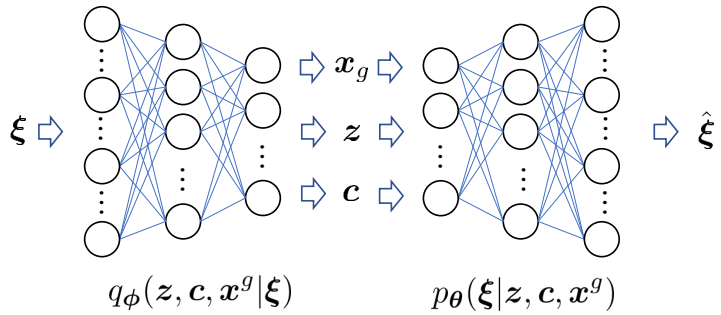


Fig. 2: Network architecture of the proposed autoencoder trajectory primitive. $\boldsymbol{z}$ is the continuous latent code and $\boldsymbol{c}$ is the discrete latent code.

supervised auxiliary code enables the generalization of the trajectories to different goal positions. We employ the objective function given by

$$\mathcal{J} = \mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{D}}\left[\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{x}_g|\boldsymbol{\xi})\right] \tag{5}$$

where

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z},\boldsymbol{c},\boldsymbol{x}^g|\boldsymbol{\xi})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{\xi}|\boldsymbol{z}, \boldsymbol{c}, \boldsymbol{x}^g)] \\ - \gamma \left| D_{\mathrm{KL}}\big(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{\xi})||p(\boldsymbol{z})\big) - C_{\boldsymbol{z}}\right| \\ - \gamma \left| D_{\mathrm{KL}}\big(q_{\boldsymbol{\phi}}(\boldsymbol{c}|\boldsymbol{\xi})||p(\boldsymbol{c})\big) - C_{\boldsymbol{c}}\right|.\end{aligned} \tag{6}$$

It can be seen that our objective function is similar to that of the M2 model proposed by Kingma et al. [23]. The second term in (5) can also be viewed as a regularization term. We use the reparametrization trick in [6] for the continuous latent variable $\boldsymbol{z}$. To learn the discrete latent variable $\boldsymbol{c}$, we employ the reparametrization with the Gumbel Max trick as in [24].

The network architecture is shown in Fig. 2. The decoder obtained in our framework can be used as a user-interface for motion planning, in which the goal position can be specified from a given task and the user can tune the latent code to obtain a preferable shape of the trajectory. We employ fully-connected neural networks with two hidden layers for both the encoder and decoder. The activation function was ReLU in our implementation. In our implementation, input and reconstructed trajectories are represented in configuration space. Therefore, inputs and outputs of the neural network are bounded between $-\pi$ to π. We think this property is suitable for training a neural network; if the value of inputs and outputs are not bounded, it would be necessary to use the batch normalization technique.

3.2 Trajectory Augmentation

When demonstrated trajectories $\{\boldsymbol{\xi}_i^{\mathrm{demo}}\}_{i=1}^M$ are given, we sample trajectories around the demonstrated trajectories to create a dataset $\mathcal{D} = \{\boldsymbol{\xi}_i\}_{i=1}^N$ with a

sufficient number of trajectories to train the neural network. This approach is inspired by the data augmentation commonly used in the computer vision community.

To obtain various goal configurations, we sample perturbation at the goal configuration by following a Gaussian distribution $\Delta \boldsymbol{q}_T \sim \mathcal{N}(\mathbf{0}, \Sigma_g)$ where Σ_g is a covariance matrix, which is diagonal in our implementation. The sampled perturbation is smoothly propagated to the whole trajectory as

$$\Delta \boldsymbol{\xi}_g = M^\dagger [0, \dots, 0, \Delta \boldsymbol{q}_T]^\top \tag{7}$$

where $M^\dagger$ is the Moore-Penrose pseudo-inverse of the matrix M defined by

$$M = \begin{bmatrix} 0 & 0 & 0 & \dots & & 0 \\ 0 & 2 & -1 & & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & \\ 0 & 0 & -1 & \ddots & 0 & 0 \\ 0 & 0 & & 2 & -1 & 0 \\ \vdots & \vdots & \ddots & -1 & 2 & -1 \\ 0 & 0 & \dots & 0 & -1 & 2 \end{bmatrix}. \tag{8}$$

The matrix M is used in the trajectory update in CHOMP [9] and is used to project the trajectory onto the constraint trajectory space as we describe later. M plays a role in smoothly propagating the difference of the trajectory to the whole trajectory.

We also sample perturbation of the whole trajectory by following the distribution given as

$$\beta_{\text{traj}}(\boldsymbol{\xi}) = \sum_{m=1}^{M} U(m) \mathcal{N}(\boldsymbol{\xi}_m^{\text{demo}}, a M^\dagger), \tag{9}$$

where a is a constant, and $U(m)$ is the uniform distribution. In prior work [25], the use of this covariance matrix for sampling trajectories was proposed, and it was empirically shown that using this covariance matrix leads to a smooth perturbation of the entire trajectory without changing the start and goal configurations. Combining these techniques, we sample trajectories by following

$$\boldsymbol{\xi}_{\text{sample}} = \Delta \boldsymbol{\xi}_g + \boldsymbol{\xi}_\beta \tag{10}$$

where $\Delta \boldsymbol{\xi}_g$ is obtained from (7), and $\boldsymbol{\xi}_\beta$ is obtained from $\beta_{\text{traj}}(\boldsymbol{\xi})$ given by (9).

3.3 Projection onto the Constraint Space

Although our decoder generates a trajectory from a given goal position, the generated trajectory does not precisely satisfy the given goal position. However, it is difficult to explicitly incorporate the constraints such as via-points

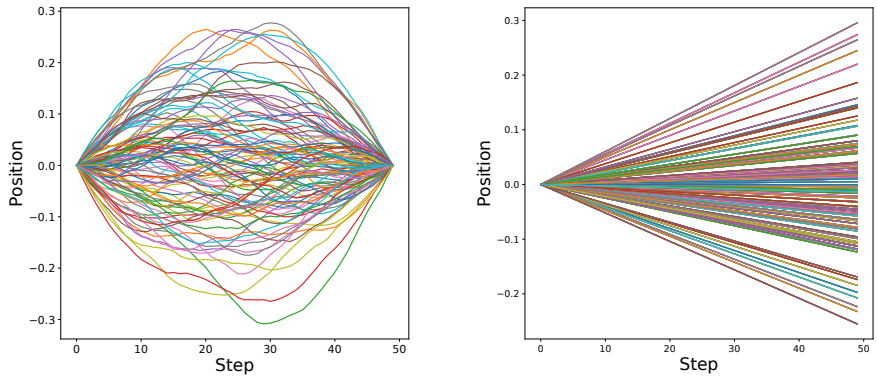


Fig. 3: Visualization of trajectories sampled for trajectory augmentation.

or joint limits in the autoencoder. To cope with this issue, we project the generated trajectory onto the constraint solution space using a trajectory optimization method. To obtain the trajectory that satisfies the constraints, we can employ trajectory optimization methods such as CHOMP [9] and TrajOpt [10]. In this study, we employ the trajectory optimization method based on CHOMP. The covariant trajectory update of CHOMP is given by

$$\boldsymbol{\xi}^{\text{new}} = \arg\min_{\boldsymbol{\xi}} \left\{ \mathcal{C}(\boldsymbol{\xi}^c) + \boldsymbol{g}^\top (\boldsymbol{\xi} - \boldsymbol{\xi}^c) + \frac{\eta}{2} \left\| \boldsymbol{\xi} - \boldsymbol{\xi}^c \right\|_M^2 \right\} \tag{11}$$

where $\boldsymbol{g} = \nabla \mathcal{C}(\boldsymbol{\xi})$,$\boldsymbol{\xi}^{\text{new}}$ is the updated plan of the trajectory, $\boldsymbol{\xi}^c$ is the current plan of the trajectory, η is a regularization constant, and $\|\boldsymbol{\xi}\|_M^2$ is the norm defined by a matrix M as $\|\boldsymbol{\xi}\|_M^2 = \boldsymbol{\xi}^\top M \boldsymbol{\xi}$. The trajectory update in (11) is equivalent to the following:

$$\boldsymbol{\xi}^{\text{new}} = \boldsymbol{\xi}^c - \frac{1}{\eta} M^{-1} \boldsymbol{g}. \tag{12}$$

When the position of the end-effector at the goal position in the current plan of the trajectory deviates from the given one, we shift the goal configuration and update the entire trajectory by iterating the following update as discussed in [26]:

$$\boldsymbol{\xi}^{\text{new}} = \boldsymbol{\xi}^c + \alpha M^{-1} [0, \dots, 0, \Delta \tilde{\boldsymbol{q}}_T]^\top, \tag{13}$$

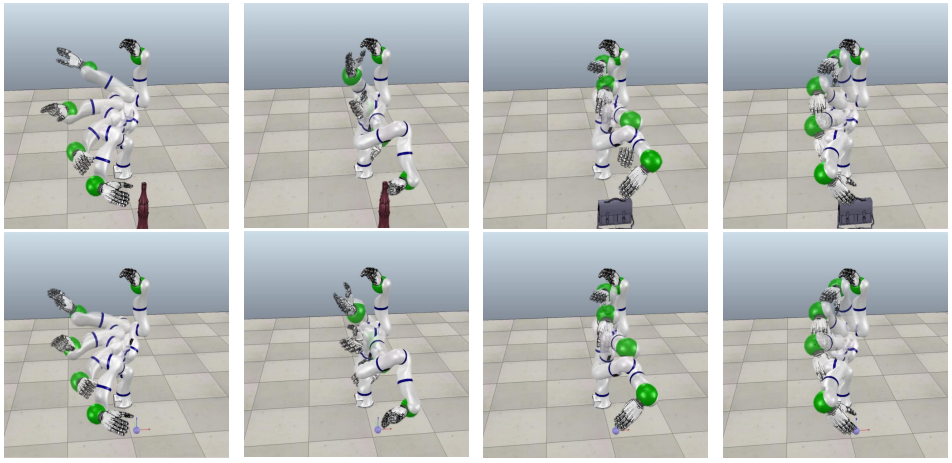
where α is the learning rate and $\Delta \tilde{\boldsymbol{q}}_T$ is given by

$$\Delta \tilde{\boldsymbol{q}}_T = J^{-1} \begin{bmatrix} \boldsymbol{x}_{\text{end}}(\boldsymbol{q}_T^0) - \boldsymbol{x}_{\text{end}}(\boldsymbol{q}_T^c) \\ \boldsymbol{0} \end{bmatrix}. \tag{14}$$

Constraints such as via-points and joint limits can also be incorporated in the same manner.

Algorithm 1 Autoencoder Trajectory Primitive (ATP)

Training phase:
Input: Trajectories demonstrated by experts $\{\boldsymbol{\xi}_i^{\text{demo}}\}_{i=1}^{M}$
Sample N trajectories by following (7) and (9)
Train the neural network by minimizing $\mathcal{J}$ in (5)
Planning phase:
Input: Goal position $\boldsymbol{x}_g$, and latent code z, c
Generate the trajectory $\boldsymbol{\xi}^*$ with the decoder
(Optional) Project the trajectory onto the constraint solution space
Return: planned trajectory $\boldsymbol{\xi}^*$



(a) (b) (c) (d)

Fig. 4: Input trajectories and reconstructed trajectories. In (a)-(d), the upper figures show the input trajectories and the lower figures show the reconstructed trajectories. The blue sphere represents the goal position in task space, i.e. $\boldsymbol{x}_g = [0, -0.7, 0.2]$. The latent variable for reconstructing the trajectory is shown in Table 2.

4 Experiments

In this experiment, four demonstrated trajectories were used. To train the ATP, we created 4000 synthetic trajectories by using the method described in Section 3.2. The dimension of discrete variable $\boldsymbol{c}$ was four, and that of the continuous variable $\boldsymbol{z}$ was five. In the following, we show the results obtained when a neural network is trained using the conditions shown in Table 1. We simulated a KUKA LWR robot[1] with 7 degrees of freedoms. A simulator is developed using V-REP [27]. To implement the variational autoencoder, we adopted the PyTorch implementation provided by the author of [8][2].

[1] We customized the color to show the motion more clearly.
[2] available at https://github.com/Schlumberger/joint-vae

Fig. 4 shows the input trajectories and reconstructed trajectories[3]. In this experiment, input trajectories are designed to represent different behavior patterns. Figs 4(a) and (b) show trajectories planned for grasping a bottle with different approach angles. Likewise, Figs 4(c) and (d) show trajectories planned for grasping a bag with different orientations. Although other frameworks often require multiple demonstrations of the same pattern, the result shows that our neural network can model multiple types of behaviors with a single model. To examine the quality of the reconstructed trajectories, the trajectories shown in Fig. 4 are not projected onto the constraint solution space. Each input trajectory is successfully reconstructed as show in Fig. 4. The latent variable generated for reconstructing the input trajectories are shown in Table 2. Interestingly, the ATP encodes trajectories shown in Fig. 4(a) and (b) into the same discrete latent code as shown in Table 2. As a result, the input trajectories shown in Fig. 4(a) and (b) are interpolated via the continuous latent variable $\boldsymbol{z}$ as we will see later. The same discussion holds for the input trajectories shown in Fig. 4(c) and (d); these trajectories are encoded in the same discrete latent code $\boldsymbol{c} = [0, 0, 0, 1]$, and they are interpolated via the latent variable $\boldsymbol{z}$. Thus, the trajectories demonstrated for the same tasks are categorized into the same class represented by the discrete latent variable $\boldsymbol{c}$ in this experiment. This result indicates that the APT can learn meaningful discrete representations in an unsupervised manner.

As discussed in [8], the KL divergence $D_{\mathrm{KL}}\big(q(\boldsymbol{z}, \boldsymbol{c}|\boldsymbol{\xi})||p(\boldsymbol{z}, \boldsymbol{c})\big)$ is the upper-bound of the mutual information between the latent variable and the data.

[3] Please note that the frames shown in the figures are not synchronized due to the limitation of our implementation.

Table 1: Conditions for training the neural network

Parameter	Value
# of epochs	250
batch size	100
# of time step in a trajectory	50
# of units in hidden layers in the encoder	(300, 200)
Activation function in the encoder	(relu, relu)
# of units in hidden layers in the decoder	(200, 300)
Activation function in the decoder	(relu, relu)

Table 2: Latent variables generated for reconstruction shown in Fig. 4

Trajectory in Fig. 4	$\boldsymbol{z}$	$\boldsymbol{c}$
(a)	[0.02, -0.25, 0.01, 0.04,-0.06]	[0., 0., 0., 1.]
(b)	[0.05, 0.57, -0.01, 0.04, -0.10]	[0., 0., 0., 1.]
(c)	[0.15, -0.30, -0.06, -0.01, -0.09]	[1., 0., 0., 0.]
(d)	[0.12, 0.28, -0.06, -0.03, -0.07]	[1., 0., 0., 0.]

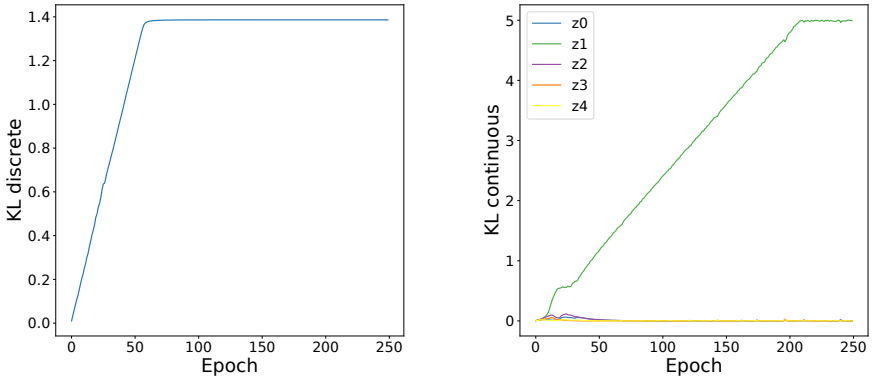


Fig. 5: The change of KL divergence during the training of our autoencoder.

Therefore, we plot the KL divergence of each latent unit during training in Fig. 5 as in [8]. As shown, among the five continuous variables, only one channel z_1 has a significant value; other four channels do not. This result indicates that we can obtain various trajectories that cover the training dataset by selecting one of four discrete variable and changing only z_1.

The trajectories obtained by changing $\boldsymbol{c}$ and z_0 are shown in Fig. 6. It is evident that the change of z_0 does not result in a significant change of the generated trajectories. This result is reasonable because the log of the KL divergence in Fig. 5 indicates that z_0 does not contain much information.

The trajectories obtained by changing $\boldsymbol{c}$ and z_1 are shown in Fig. 7. It is clear that the change of z_1 results in significant change of the shapes of generated trajectories. This result also fits with the result that only z_1 contains significant information as shown in Fig. 5. In the top row of Fig. 7, which corresponds to $\boldsymbol{c} = [1, 0, 0, 0]$, one can see that the input trajectories shown in Fig. 4(a) and (b) are continuously interpolated via the latent variable $\boldsymbol{z}$. Accordingly, the trajectory shown in the middle of the top row of Fig. 7 appears to be a mixture of the trajectories shown in Fig. 4(a) and (b). Likewise, it can be seen in the bottom row of Fig. 7, which corresponds to $\boldsymbol{c} = [0, 0, 0, 1]$, that the input trajectories shown in Fig. 4(c) and (d) are continuously interpolated via the latent variable $\boldsymbol{z}$. As a result, the middle of the bottom row of Fig. 7 appears to be a mixture of the trajectories shown in Fig. 4(c) and (d).

The trajectories obtained by changing $\boldsymbol{x}_g$ are shown in Fig. 8. The trajectories are properly generalized to different goal positions of the end-effector. As discussed in [28], the output of the variational autoencoder is amortized variational inference. Therefore, the generalization with the decoder is not accurate, e.g. the positioning error is approximately from 0.01 to 0.05 m. However, after the projection with CHOMP, the positioning error at the goal position can be

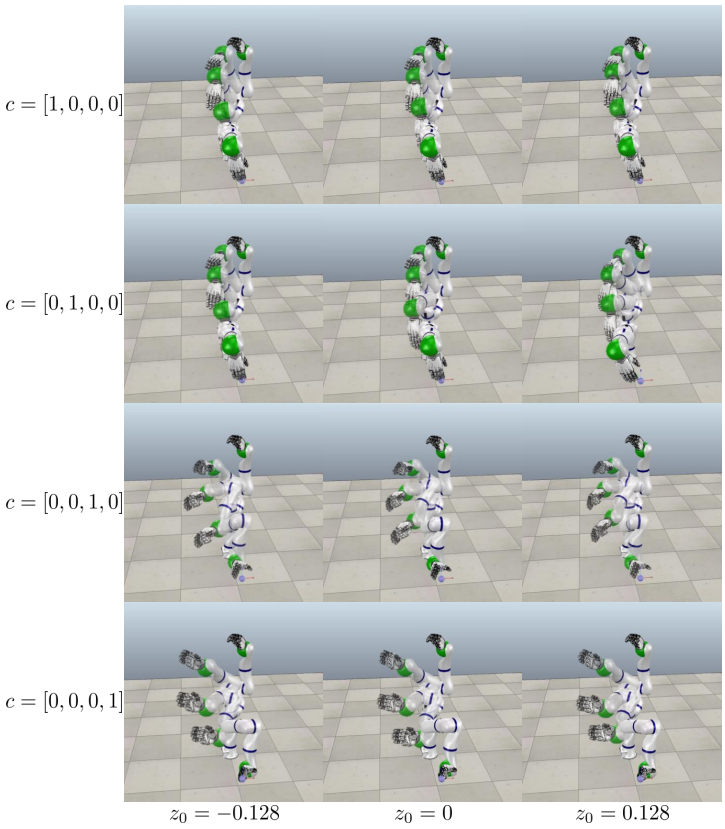


Fig. 6: Trajectories generated by changing z_0. From the top, each row corresponds to $\boldsymbol{c} = [1, 0, 0, 0]$, $\boldsymbol{c} = [0, 1, 0, 0]$, $\boldsymbol{c} = [0, 0, 1, 0]$, $\boldsymbol{c} = [0, 0, 0, 1]$, respectively. $z_1 = z_2 = z_3 = 0$ and $\boldsymbol{x}_g = [0, -0.7, 0.2]$ in all trajectories. The column corresponds to varying z_0.

significantly small, e.g., less than 1mm. This results show that the ATP can generalize the learned trajectories to different goal positions.

From these results, it is evident that our autoencoder learned interpretable latent representations from a limited number of demonstrated trajectories. On the other hand, the hand orientation and the shape of the entire trajectory is entangled in the learned latent variable; when changing z_1, both the orientation and the shape of the entire trajectory change. In practice, a user may need to tune the shape of the entire trajectory without changing the hand orientation, e.g. for collision avoidance. However, when using the latent variable learned in this study, it is not possible to change the shape of the entire trajectory without changing the hand orientation. We think that this result is due to the property of the objective function of our neural network and that this point needs to be addressed in future work.

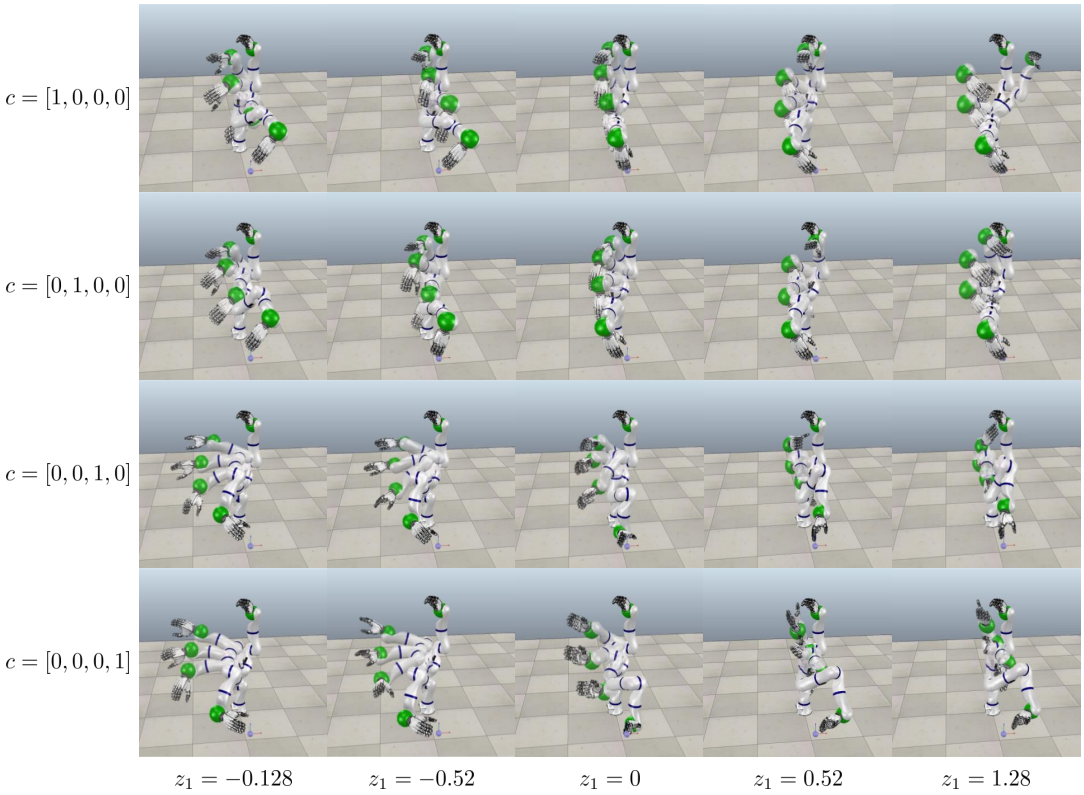
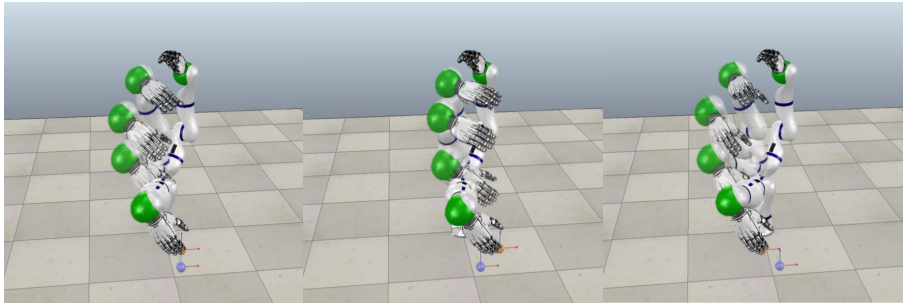


Fig. 7: Trajectories generated by changing z_1. From the top, each row corresponds to $\boldsymbol{c} = [1, 0, 0, 0]$, $\boldsymbol{c} = [0, 1, 0, 0]$, $\boldsymbol{c} = [0, 0, 1, 0]$, $\boldsymbol{c} = [0, 0, 0, 1]$, respectively. $z_0 = z_2 = z_3 = 0$ and $\boldsymbol{x}_g = [0, -0.7, 0.2]$ in all trajectories. The column corresponds to varying z_1.

5 Discussion

The results indicate that the ATP learns useful latent representations for tuning the trajectory shape. The learned decoder can be used as a motion planner that takes the goal position and latent variable as its input. Using the ATP, a user can teach the desired motion by performing a handful of demonstrations, and when a trajectory is planned for a new task, a user can also select and modify the trajectory by changing the discrete and continuous latent variables. The trajectory augmentation presented in this paper can be viewed as a way to autonomously explore the trajectory for learning useful low-dimensional representations.

Existing movement primitive frameworks, such as DMP and ProMP, often require separate models for representing diverse behaviors. In other words, these frameworks require label information for learning multiple types of be-



$\boldsymbol{x}_g = [0.0, -0.7, 0.3]$ $\boldsymbol{x}_g = [0.1, -0.7, 0.3]$ $\boldsymbol{x}_g = [-0.1, 0.7, 0.3]$

Fig. 8: Trajectories generated by changing $\boldsymbol{x}_g$. The trajectories are properly generalized to different goal positions. $\boldsymbol{c} = [1, 0, 0, 0]$ and $\boldsymbol{z} = [0., 1.28, 0., 0., 0.]$ in all trajectories. The orange sphere indicates the specified goal position, and the blue sphere indicates the goal position used in the demonstrated trajectories. The unit of $\boldsymbol{x}_g$ is the meter in this simulation.

haviors. As a result, if a given demonstration dataset contains multiple types of behaviors without labels, they often fails to model the given trajectories. However, as shown by the experimental results, the ATP can model diverse behavior by encoding the trajectory types into the latent variables in an unsupervised manner. This property of the ATP is beneficial in practice, since it will reduce the effort to consider the types of demonstrated trajectories. Prior work such as [4] considered the combination of movement primitives by introducing the activation factor. In contrast, ATP can deal with the blending of multiple types of behaviors through the continuous latent variable, which does not require learning multiple models. This property is also useful in practice when the user need to plan a trajectory which is similar to input trajectories but different from them.

Neural networks are often considered to be incomprehensible due to their complexity. However, our work shows that neural networks can provide a way to interpret and manipulate incomprehensible high-dimensional data by finding low-dimensional latent representations. Although trajectory planning in robotic systems often require expert knowledge, the low-dimensional representation found by our framework can be used to tune the trajectory shape without expert knowledge.

We think that the proposed framework can be used for task-level motion planning. The task-level motion planning is often built on pre-fixed motion planner, and it is not trivial to tune each lower-level planner. However, by using our framework, lower-level trajectories can be represented by low-dimensional information and the tuning of such a low-dimensional vector should be much easier compared to the raw representations of trajectories.

6 Conclusions

We proposed the autoencoder trajectory primitive (ATP), which is a framework for modeling demonstrated trajectories with a neural network. In the proposed framework, the latent variables that encodes the multiple types of trajectories are learned in an unsupervised manner. The trajectory augmentation trick was proposed to address the issue of the size of the training data. The learned decoder can be used as a motion planner in which the user can specify the goal position and the trajectory types by setting the latent variables. Our experimental results show that a neural network can be trained with a handful of demonstrated trajectories and that the ATP successfully learns discrete and continuous low-dimensional latent variables.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.
2. A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Advances in Neural Information Processing Systems (NIPS)*, 2002.
3. A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," in *Neural computation*, vol. 25, no. 2, 328–373, 2013.
4. A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic movement primitives," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*. mit press, 2013.
5. Y. Huang, L. Rozo, and J. S. andand D.G. Caldwell, "Kernelized movement primitives," *International Journal of Robotics Research*, 2019.
6. D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.
7. I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems (NIPS)*, 2014.
8. E. Dupont, "Learning disentangled joint continuous and discrete representations," in *Advances in Neural Information Processing Systems 31 (NIPS 2018))*, 2018.
9. M. Zucker, N. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. A. Bagnell, and S. Srinivasa, "Chomp: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, pp. 1164–1193, 2013.
10. J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.
11. L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars., "Probabilistic roadmaps for path planning in high-dimensional conguration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, 1996.
12. S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, 2001.

13. T. Osa, N. Sugita, and M. Mitsuishi, "Online trajectory planning and force control for automation of surgical tasks," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 675–691, April 2018.
14. A. Shon, K. Grochow, A. Hertzmann, and R. P. Rao, "Learning shared latent structure for image synthesis and robotic imitation," in *Advances in Neural Information Processing Systems (NIPS)*, 2005.
15. D. B. Grimes and R. P. Rao, "Learning nonparametric policies by imitation," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008.
16. N. Lawrence, "Probabilistic non-linear principal component analysis with gaussian process latent variable models," *Journal of Machine Learning Research*, vol. 6, pp. 1783–1816, 2005.
17. S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.
18. T. Haarnoja, V. H. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine, "Composable deep reinforcement learning for robotic manipulation," in *Proceedings of the IEEE Conference on Robotics and Automation (ICRA)*, 2018.
19. J. Merel, L. Hasenclever, A. Galashov, A. Ahuja, V. Pham, Y. W. T. G. Wayne and, and N. Heess, "Neural probabilistic motor primitives for humanoid control," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.
20. S. Levine, "Reinforcement learning and control as probabilistic inference: Tutorial and review," *arXiv*, 2018.
21. S. Arnold and K. Yamazaki, "Fast and flexible multi-step cloth manipulation planning using an encode-manipulate-decode network (em*d net)," *Frontiers Neurorobotics*, vol. 13.
22. I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.
23. D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, "Semi-supervised learning with deep generative models," in *Advances in Neural Information Processing Systems (NIPS)*, 2014.
24. C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.
25. M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2011, pp. 4569–4574.
26. A. D. Dragan, K. Muelling, J. A. Bagnell, and S. S. Srinivasa, "Movement primitives via optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 2339–2346.
27. E. Rohmer, S. P. N. Signgh, and M. Freese, "V-rep: a versatile and scalable robot simulation framework," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.
28. C. Cremer, X. Li, and D. Duvenaud, "Inference Suboptimality in Variational Autoencoders" in *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.