Movement Primitive Learning and Generalization using Mixture Density Networks

You Zhou, Jianfeng Gao and Tamim Asfour

***Abstract*— Representing robot skills as movement primitives that can be learned from human demonstration and adapted to new tasks and situations is a promising approach towards intuitive robot programming. To allow such adaptation, a mapping between task parameters and movement primitives parameters is needed, and different approaches have been proposed in the literature to learn such a mapping. In human demonstrations, however, multiple modes and models exist, which should be taken into account when learning these mappings and generalized movement primitive representations. Here, a challenging problem is mode or model collapse. In order to solve this problem, we propose using a Mixture Density Network (MDN) that takes task parameters as input and provides a Gaussian Mixture Model (GMM) of the movement primitive parameters. To avoid mode and model collapse during MDN training, we introduce an entropy cost to achieve a more balanced association of demonstrations to GMM mixture components. Since it is often easier to collect failed examples by using an underfitted MDN model instead of additional human demonstrations, we introduce a failure cost to reduce the occurrence of failures in future executions. We evaluated our approach in simulation and real robot experiments and showed that the method outperforms previous approaches.**

I. INTRODUCTION

Over the past decades, robotic researchers have developed different approaches, which enable robots to learn from humans, imitate human behavior, and autonomously improve their movements. As a replacement for manual robot programming, learning from human demonstrations, also called *imitation learning*, has been proven a promising and powerful technique for intuitive robot programming [1]. Inspired by neuro-psychological finding [2], we use movement primitives (MP) as essential building blocks for describing robot motions. In this context, the question of how a generalizable and compact representation of movement primitives can be learned from demonstrations and adapted to new situations and changing task parameters is an active research area in robotics. Different movement primitive representations have been proposed such as Dynamic Movement Primitive (DMP) ([3]), Probabilistic Movement Primitive (ProMP) ([4]) and Task-Parameterized Gaussian Mixture Model (TP-GMM) ([5]). These representations can adapt to task parameters in the space in which we define these primitives. For example, DMP adapts to a new start or goal, and ProMP adapts to intermediate via-points. TP-GMM adapts to changes of predefined local frames, where we describe the demonstrated trajectories. However, for different tasks, the task parameters could have different meanings and are not directly associated with spatial or temporal requirements for motion trajectories. For example, a robot throwing a ball to a target specified in the Cartesian space by executing a motion learned and represented in the robot's joint space should be able to adapt the learned motion to new targets. None of these approaches above can adapt a learned movement primitive to new targets specified in a different space.

The authors are with the Institute for Anthropomatics and Robotics, Karlsruhe Institute of Technology, Karlsruhe, Germany. {you.zhou, jianfeng.gao, asfour}@kit.edu

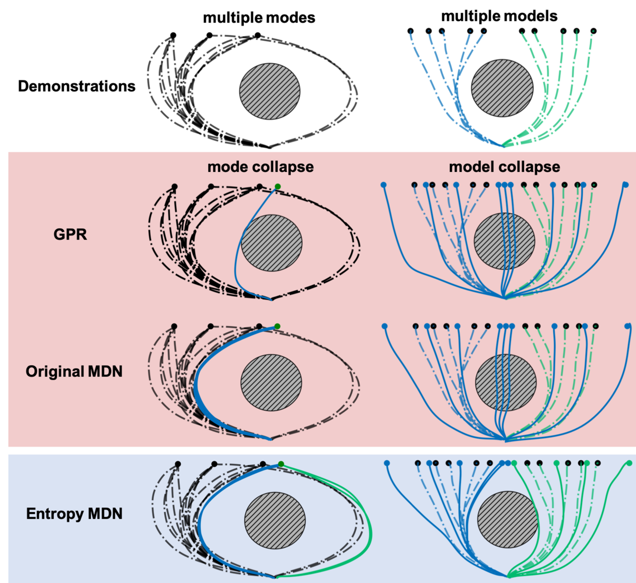

Fig. 1: Solving the mode and model collapse problem in the obstacle avoidance problem using the entropy Mixture Density Networks. The dashed curves denote the demonstrations. The solid curves denote the generated trajectories.

For such generalization problems, methods have been proposed which learn a task-specific mapping between task and movement primitive parameters using regression models such as Locally Weighted Regression ([6], [7]), Gaussian Process Regression ([8]) or Deep Neural Networks ([9])

However, such methods cannot deal with the problem of *mode* and *model collapse*. Given the task of reaching a target while avoiding an obstacle (see Figure 1), the goal can be reached using two different modes, i.e., by trajectories passing the obstacle from the left or the right. Thus, learning a movement primitive for such a task should take multiple modes in human demonstrations into account to increase the diversity of motions, which is beneficial in the case of changing task constraints. Even though there are no multiple modes for each single task parameter query in human demonstrations, there might exist multiple models for different types of task parameters. As shown in Figure 1 (right column), the goals on the left and right sides of the obstacle are reached separately by two types of trajectories resulting from two different models to allow passing the obstacle from the left or the right.

In many applications, human demonstrations might use multiple modes and models at the same time. To avoid both mode and model collapses, we propose to learn a mapping from the task parameter query $\boldsymbol{q}$ to a Gaussian Mixture Model (GMM) of the movement primitive (MP) parameters $\boldsymbol{w}$ with Mixture Density Networks (MDN). Each mixture component of the GMM represents either a mode for one specific task parameter query or a model for multiple task parameter queries. However, training MDN with only the Negative-Log-Likelihood (NLL) cost, as usually done in the original MDN, might still lead to mode and model collapses, especially when the set of demonstrations is relatively small as shown in Figure 1. In order to further reduce the occurrence of both collapses, we introduce an entropy cost function for training MDN (entropy MDN).

The left column of Figure 1 shows the human demonstrations (dashed curves) for the obstacle avoidance with an imbalanced distribution of examples associated with the two different modes, as well as the results obtained with different approaches for a new task parameter query (the green dot). As can be seen, our approach (entropy MDN) generates multiple solutions (here: 5 solutions in Figure 1) that cover both demonstration modes (blue and green) for the same task parameter query. We achieve this result with the entropy cost function that ensures a more balanced probability associated with the different modes. The original MDN generates multiple solutions, but these solutions (also 5) reflect only one mode in the human demonstrations since it associates a high probability for the dominant mode. GPR generates only one single solution that is close to the examples for the dominant mode.

The right column of Figure 1 shows the demonstrations (blue and green dashed curves) associated with two different models and the trajectories generated by different approaches (solid curves) for several task parameter queries (the colorized dots). As shown in the right bottom figure, our approach (entropy MDN) successfully learns the two models and generates the solution for the task parameter query correspondingly. We achieve this result also with the entropy cost function that ensures that each model is associated with some human demonstrations. The original MDN has a higher chance of being stuck in the local minima of the NLL, where only one model is trained and overfitted for all task parameter queries. This fact leads to the failure of the task execution, i.e., the collision of the obstacle, especially when the task parameter queries are on the boundary of two models. GPR can only learn one model; thus, it suffers the same problem as the original MDN.

For many tasks, it is often easier to collect failed samples with an underfitted MDN model instead of requesting additional human demonstrations. In order to further improve the MDN performance, we introduce a failure cost function that reduces the occurrence of the same failures for a given task parameter query.

II. Related Works

Dynamic Movement Primitive (DMP) is one of the popular approaches to represent robot motions. A DMP consists of a damped spring system and a non-linear force term $f(x)$ such that

$$\begin{aligned} \tau\dot{v} &= K(g-y) - Dv + (g-y_0)f(x)x, \\ \tau\dot{y} &= v, \\ f(x) &= \frac{\sum_{i=1}^{N} \boldsymbol{\psi}_i(x)\boldsymbol{w}_i}{\sum_{i=1}^{N} \boldsymbol{\psi}_i(x)}. \end{aligned} \tag{1}$$

where v, y are the scaled velocity and position of the trajectory point. g, y_0, τ, as the hyper-parameters, represent goal, start and temporal factor separately. x is the canonical variable which goes from 1 to 0. The force term $f(x)$ is a linear regression model with N squared exponential kernels (SEKs) $\boldsymbol{\psi}_i(x) = exp(-h_i(x-c_i)^2)$, where h_i, c_i are fixed constants. $\boldsymbol{w} = (\boldsymbol{w}_1, ..., \boldsymbol{w}_N)^T$ is a vector representing the DMP parameters. DMP generalization means replacing $\boldsymbol{w}$ with a parameterized function $\omega(\boldsymbol{q})$. As mentioned before, this function can be represented and learned with different approaches such as Gaussian Process Regression (GPR) ([8]), Locally Weighted Regression (LWR) ([6], [7]), Support Vector Regression (SVR) ([10]) or Deep Neural Networks ([9]). To train those models, the training dataset is collected as M pairs of task parameter queries and MP parameters $\{(\boldsymbol{q}_i, \boldsymbol{w}_i)\}_{i=1}^{M}$, where the i-th MP parameter $\boldsymbol{w}_i$ is learned from the i-th demonstration and corresponding to the i-th task parameter $\boldsymbol{q}_i$.

Those methods require two parameterized functions $f(x)$ and $\omega(\boldsymbol{q})$. The output of the $\omega(\boldsymbol{q})$ is the parameter $\boldsymbol{w}$ of the $f(x)$. Hence, they are called two-steps methods. In [11], the authors proposed to combine two functions to one single function $f(x, \boldsymbol{q})$, which was learned with LWR or GPR and extended to GMM in [12]. Since these methods use only one single function, they are called one-step methods, which are more compact than the two-steps methods.

The idea of Task-Parameterized Gaussian Mixture Model (TP-GMM), suggested in [5], is to observe human demonstrations from multiple perspectives (local frames). A global GMM represents the trajectory points in a global frame and maximizes the likelihood of demonstrations from different perspectives. With the transformation of the local frames, TP-GMM achieves a better extrapolation performance than other methods. In [13], the authors extended TP-GMM and learned the sensory data together with the motion trajectories. As mentioned before, however, the task parameters considered by TP-GMM are limited.

Probabilistic Movement Primitive (ProMP) uses a linear regression model with kernel functions $\boldsymbol{\psi}(\cdot)$ to directly represent the motion trajectory $y(x) = \boldsymbol{\psi}(x)^T\boldsymbol{w}$. In [4], the authors assumed that $\boldsymbol{w}$ follows a Gaussian distribution. In [14], the Gaussian distribution was extended to a GMM. For human-robot interactions, the ProMP parameter ($\boldsymbol{w} = \{\boldsymbol{w}_o, \boldsymbol{w}_c\}$) is separated to encode both human ($\boldsymbol{w}_o$) and robot

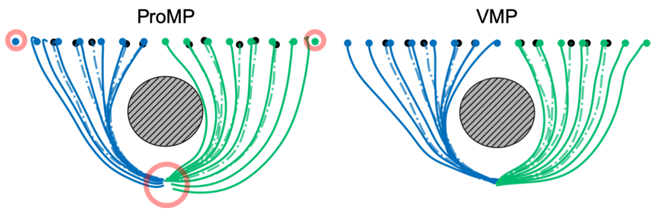

Fig. 2: Comparison of ProMPs and VMPs: The dashed curves are demonstrations, and the solid curves are generated trajectories for different goals. The red circles indicate the starts and goals missed by ProMP.

motions ($\boldsymbol{w}_c$). With the conditional probability, the robot MP parameter $\boldsymbol{w}_c$ is inferred based on the human MP parameter $\boldsymbol{w}_o$. Both methods in [13] and [14] learn a generative model and use the conditional probability to infer the unknown variables based on the known ones.

In this paper, we use Via-points Movement Primitive (VMP), a movement primitive formulation presented in our previous work in [15] and described in subsection III-A. Instead of learning a generative model, we learn an MDN to directly map from a task parameter $\boldsymbol{q}$ to a GMM of the MP parameters $\boldsymbol{w}$. As GPR or SVR for DMP, our method belongs to the two-steps methods.

III. Movement Primitive Generalization

A. Via-points Movement Primitive

We use Via-points Movement Primitive (VMP) (see [15]) to represent robot motions. VMP consists of an elementary trajectory h and a shape modulation f as follows:

$$y(x) = h(x) + f(x) = g + x(y_0 - g) + \boldsymbol{\psi}(x)^T \boldsymbol{w}, \quad (2)$$

where x is the canonical variable, which goes from 1 to 0 with a linear decay canonical system. The elementary trajectory takes the form of a polynomial. Here, it is a first-oder polynomial, i.e. a line connecting the start and the goal. By changing the y_0, g, as the hyper-parameters, VMP adapts to the new start and goal. As the non-linear force term in DMP, the shape modulation is a linear regression model with the SEKs $\boldsymbol{\psi}$. $\boldsymbol{w}$ is the parameter vector of VMP.

In [15], we assume that the parameter $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ follows a Gaussian distribution and the Maximum Likelihood Estimation (MLE) of $\boldsymbol{\mu}$ is the empirical mean of all $\boldsymbol{w}$s, each of which is corresponding to one demonstration and obtained by solving a least square problem.

Here, we extend the Gaussian distribution to a GMM and consider task parameter queries $\boldsymbol{q}$ as inputs:

$$\boldsymbol{w} = \omega(\boldsymbol{q}) \sim \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{\mu}_k(\boldsymbol{q}), \boldsymbol{\Sigma}_k(\boldsymbol{q}))^{z_k}, \quad (3)$$

where z_k is an element of a K-dimensional binary variable $\boldsymbol{z} = (z_1, z_2, ..., z_K)^T$ where only one particular element is equal to 1 and all other elements are zero. The probability of the k-th element of $\boldsymbol{z}$ being equal to 1 is

$$p(z_k = 1) = \pi_k(\boldsymbol{q}).$$

The probability of the result $\boldsymbol{w}$ given the task parameter $\boldsymbol{q}$ is

$$p(\boldsymbol{w}|\boldsymbol{q}) = \sum_{k=1}^{K} \pi_k(\boldsymbol{q}) \mathcal{N}(\boldsymbol{\mu}_k(\boldsymbol{q}), \boldsymbol{\Sigma}_k(\boldsymbol{q})) \quad (4)$$

We assume that the number of the mixture components K of the GMM is known. And $\{\pi_k(.), \boldsymbol{\mu}_k(.), \boldsymbol{\Sigma}_k(.)\}_{k=1}^{K}$ are the functions to be learned.

The advantage of VMP over DMP is its ability to adapt to intermediate via-points by modifying the elementary trajectory $h(x)$ (see [15]). Apart from the via-points adaptation, extending force term in a DMP to a GMM makes VMPs and DMPs exchangeable. Since ProMP lacks the elementary trajectory and has no hyperparameters y_0, g, the ProMP parameter function $\boldsymbol{w} = \omega(\boldsymbol{q})$ determines both the motion trajectory shape and its start and goal. In many tasks, the start and goal are a part of the task parameter queries. With VMP, we reduce the learning complexity, because one requirement of the task, i.e., reaching a new goal, is directly satisfied with the hyperparameter g. For example, in Figure 2, in contrast to VMP, not all the trajectories generated by ProMP reach the goal if we use the entropy MDN and select the most probable parameter $\boldsymbol{w}$ from the output distribution. In some tasks, however, the goal is not a part of the task parameters and is necessary for the task execution. For these tasks, VMP requires learning an additional mapping from the task parameters to the goals, while ProMP provides a more compact solution. As shown in Figure 2, the entropy MDN suggested for the VMP generalization also works for the ProMP generalization.

In general, for MP generalization, M human demonstrations for different task parameter queries are collected. The purpose is to learn an MDN ($\omega(\cdot)$) mapping from the task parameter query $\boldsymbol{q}$ to the parameter distribution of the MP parameter $\boldsymbol{w}$. For MDN, we have an assumption that the number of the mixture components K of the output GMM is known.

B. Mixture Density Network

Using neural networks to learn the functions in Equation 4 results in a Mixture Density Network (MDN) (see [16]). The mixing coefficients $\pi_k(\cdot)$, the mean $\boldsymbol{\mu}(\cdot)$ and the covariance $\boldsymbol{\Sigma}(\cdot)$ are represented by the network branches as shown in Figure 3. These network branches share a common network part (blue box in Figure 3), which allows extracting common latent features.

For the covariance output, we assume that it is a diagonal matrix $\boldsymbol{\Sigma}$. According to [17], a GMM with a diagonal variance matrix approximates any given density function to arbitrary accuracy. The diagonal covariance output has the same dimension as the mean output. For K components, MDN has K pairs of mean and variance outputs. Each pair corresponds to a mixture component of a GMM.

The dimension of the MP parameters, as the output of MDN, determines the accuracy of the trajectory representation. With more SEKs, the MP represents the motion in a more accurate way. In the following experiments, we use

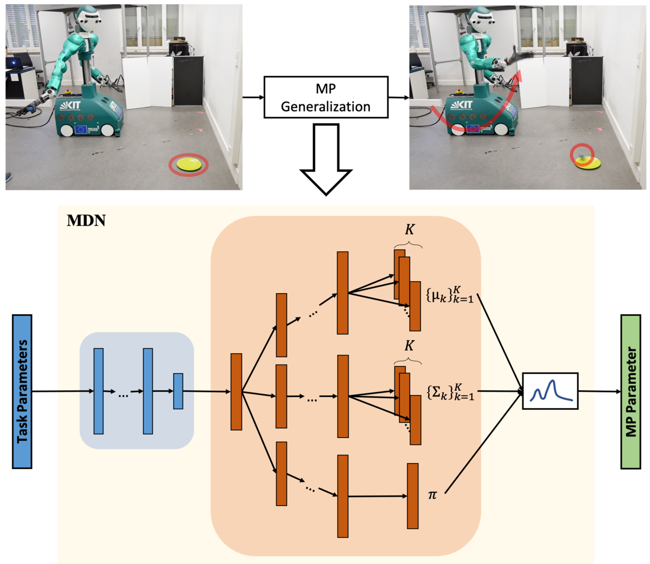

Fig. 3: The Mixture Density Network is proposed for the movement primitive generalization. As an example, with the target as the task parameter indicated by the red circle, the system generates a movement primitive parameter corresponding to a motion throwing the ball (see the red curves) on the target.

10 SEKs for each dimension. As an example, the output mean vector has 30 dimensions for a 3-dimensional motion trajectory, and there are total $K + K \times 30 \times 2$ values in the MDN output.

For one single demonstration, we calculate $\boldsymbol{w}$ by solving the least square problem for Equation 2. With M demonstrations, a training dataset $\{(\boldsymbol{q}_i, \boldsymbol{w}_i)\}_{i=1}^{M}$ is collected. The NLL is written as

$$l_{NLL}(\boldsymbol{\Theta}) = -\sum_{i=1}^{M} log\Bigg(\sum_{k=1}^{K} \pi_k(\boldsymbol{q}_i; \boldsymbol{\Theta}) \\ \mathcal{N}\Big(\boldsymbol{w}_i; \boldsymbol{\mu}(\boldsymbol{q}_i; \boldsymbol{\Theta}), \boldsymbol{\Sigma}(\boldsymbol{q}_i; \boldsymbol{\Theta})\Big)\Bigg), \tag{5}$$

where $\boldsymbol{\Theta}$ is the parameters of the network and

$$\mathcal{N}\left(\boldsymbol{w}_i; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}_i|^{1/2}} \cdot \\ exp\left(-\frac{1}{2}\sum_{j=1}^{d}\frac{(w_{i,j} - \mu_{i,j})^2}{\sigma_{i,j}^2}\right), \tag{6}$$

where d is the dimension of the output, $\boldsymbol{\mu}_i = \boldsymbol{\mu}(\boldsymbol{q}_i; \boldsymbol{\Theta})$ and $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}(\boldsymbol{q}_i; \boldsymbol{\Theta})$. A stochastic gradient descent method is used to minimize the NLL.

According to the previous works ([18], [19]), training MDN with the NLL suffers the mode collapse. In [18], to avoid the mode collapse and reduce the learning complexity, the authors suggested to fix the mean and variance on a grid defined in the output space, and only train the model that outputs the mixing coefficients to reduce the NLL. If there are enough components regularly distributed in the output space, fixed means and variances do not reduce the representation capability. However, for large dimensional outputs such as MP parameters, this method leads to a sizeable intractable grid.

In [19], the authors used MDN to predict the distribution of future car positions based on the current car position. In order to avoid the mode collapse, they separated the MDN into two parts: a sampling and an inference network. The sampling network takes the current car position as input and outputs a fixed number of hypotheses for future car positions. They train the sampling network to place hypotheses to cover all the observed outputs diversely. Based on these hypotheses, an inference network infers the parameters of the GMM. The MDN is a combination of the sampling and inference networks. The proposed method avoids the mode collapse for the car positions prediction. However, for a high dimensional output such as MP parameters, the sampling network requires a large output dimension. Hence, it is difficult to apply both methods to our problem.

C. Entropy Costs for MDN

Before introducing the entropy cost function, we first inspect the reasons why mode and model collapses occur when learning MDN from demonstrations.

The mode collapse occurs when the demonstrations associated with different modes for one task parameter query are very imbalanced. As an example, in Figure 1 (left column), only a small number of demonstrations take the path from the right side. By maximizing the likelihood of all demonstrations, MDN tends to output a small mixing coefficient for the mixture component, which corresponds to the mode with less associated training data. In theory, it is correct to associate a small probability to the event that rarely happens in the observations. However, the reasons for the imbalance of the demonstrations in different modes, such as the habit of the demonstrator, can be meaningless for correct motion generation. It is often the case that we cannot collect enough demonstrations to cover all modes. Even if there are only a few demonstrations, where the human accomplishes the task with a particular type of motions, the robot should learn these motions to increase motion diversity.

The model collapse occurs when there are only a relatively small number of demonstrations. Several mixture components of MDN, which are represented by neural networks, are powerful enough to overfit all demonstrations. After training of the MDN, instead of all K mixture components, it uses only a subset of them, which corresponds to the local minima of the NLL and results in poor performance of the MDN for some task parameter queries. As shown in Figure 1 (right column), one of the two models disappears with the original MDN, and MDN performs similar to GPR. Compared to the mode collapse, the model collapse is more severe because it can lead to the failure of the task execution.

In order to reduce the occurrence of the mode and model collapses, we introduce the negative model entropy cost function as follows:

$$l_{model}(\mathbf{\Theta}) = \sum_{k=1}^{K} p(m=k|\boldsymbol{D};\mathbf{\Theta}) \log p(m=k|\boldsymbol{D};\mathbf{\Theta}), \quad (7)$$

where

$$p(m=k|\boldsymbol{D};\mathbf{\Theta}) = \sum_{i=1}^{M} \pi_k(\boldsymbol{q}_i;\mathbf{\Theta}) p(\boldsymbol{q}_i), \quad (8)$$

and m is the component index and $p(\boldsymbol{q}_i) \propto M^{-1}$. By minimizing the cost, we increase the uncertainty of the model labels when considering all demonstrations $\boldsymbol{D}$. A high uncertainty of the model labels is equivalent to either equally distributed mixing coefficients for each task parameter query or equally distributed demonstrations to different models. In the former case, if all mixing coefficients for one specific task parameter query are almost equal and close to $1/K$, each mode has the same probability of being selected to generate motions. Hence, the mode collapse does not occur. In the latter case, if each model is associated with some demonstrations, the corresponding mixture component, i.e., the network branch, is well trained. Hence, the model collapse does not occur.

The objective function for training the entropy MDN is a weighted sum of the NLL and the entropy cost function: $w_{NLL} l_{NLL} + w_{model} l_{model}$. In the following experiments, the weights are empirically determined: $w_{NLL} = 1, w_{model} = 50$.

D. Improve MDN with Failures

In many applications, the failed samples are easily collected with an underfitted MDN model. To reduce the occurrence of these failed MP parameters for similar task parameter queries, we introduce the failure cost function as follows:

$$l_{neg}(\mathbf{\Theta}) = \sum_{i=1}^{M^*} log\left(\sum_{k=1}^{K} \pi_k(\boldsymbol{q}_i;\mathbf{\Theta}) \mathcal{N}\left(\boldsymbol{w}_i^*; \boldsymbol{\mu}_i, \mathbf{\Sigma}_{neg}\right)\right), \quad (9)$$

where the normal distribution has the same form as Equation 6 but $\mathbf{\Sigma}_{neg} = \sigma_{neg}\boldsymbol{I}$. By minimizing this cost, the output mean vector $\boldsymbol{\mu}_i$ for a specific task parameter $\boldsymbol{q}_i$ is kept away from the failed MP parameters $\{\boldsymbol{w}_i^*\}_{i=1}^{M^*}$.

If σ_{neg} is too small, the failure cost will not affect the results. On the other hand, a too big σ_{neg} can result in trajectories, which are significantly different from demonstrations. Here, we determined σ_{neg} empirically with the smallest variance of all MP parameter components.

For training an MDN with the failure cost, we prepare an evaluation dataset. After a certain number of training steps, we run the MDN on this evaluation dataset and collect the failed samples in a failures dataset $\{\boldsymbol{w}_i^*\}_{i=1}^{M^*}$. In the next training steps, we calculate the failure cost function based on the failures dataset. In order not to increase the computational cost, we use a fixed dataset size M^* and remove the earliest failed samples from it when new samples are collected.

For the evaluation of the MP generalization methods, we check whether the generated MPs accomplish the tasks with different task parameters.

In the learning from demonstrations, a successful task execution means that the generated motions are similar to the demonstrations and can accomplish the task with specific task parameters. In the proposed method, we meet the above requirement by training MDN with the NLL, which is related to the similarity between the collected and generated MP parameters. To check whether the trained model meets the latter requirement, we evaluate it with the success rate of the task execution.

For the task execution, we can only execute one motion after another. Hence, the MP parameter for the task must be determined based on the MDN output distribution in a subsequent step.

E. Generating Motions with MDN

The purpose is to generate single motions for some task parameter queries. In the following experiments, we consider two strategies: selecting the most probable mode or selecting the best one from multiple samples.

The most probable mode is the output mean vector of the mixture component that has the most significant mixing coefficient. If the Gaussian components of the output GMM have separated means, the most probable mode corresponds to the mode of GMM.

For one specific task parameter query $\boldsymbol{q}$, MDN outputs K Gaussian mixture components with their mixing coefficients $\{\pi_k\}_{k=1}^{K}$. The K modes of these Gaussian mixture components correspond to K most probable motions of different types. However, not all these K modes can accomplish the task. The most significant mixing coefficient indicates the mode that most likely succeeds. With this strategy, the MDN serves as a deterministic model. Hence, we can compare it with previous deterministic methods. Selecting the most probable mode is the simplest way to generate motion from the MDN output. Moreover, this strategy works quite well in many tasks.

However, with the most probable mode, we ignore the information provided by the output variance $\mathbf{\Sigma}$ of the MDN. Each of its diagonal elements indicates how various the generated trajectories can be at the corresponding time for successful task execution. When we draw samples from the output GMM for a specific task parameter query, the variance matrix ensures that the samples have a high probability of success. In some tasks, the most probable mode does not work very well, such as in the experiment described in subsection IV-C. To improve the performance, we draw several MP parameters from the output distribution and execute one after another for the task until success. In this case, the success rate is also dependent on the number of samples.

Moreover, with the former strategy, the MDN always generates the same motion for one specific task parameter query. In order to demonstrate motion diversity and the fact that the MDN learns the multiple modes, we also need to draw multiple samples from the output distribution (see subsection IV-D).

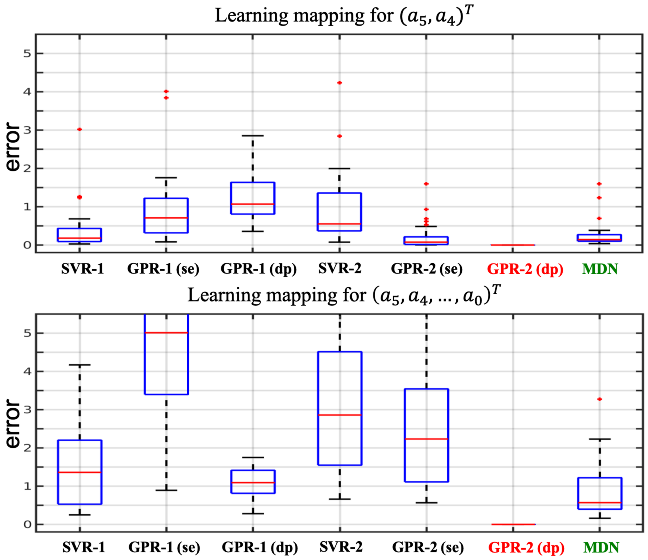

Fig. 4: The models are learned to fit a 5-th order polynomial. **Top**: the results for learning a mapping from a part of the coefficients to the MP parameters. **Bottom**: the results for learning a mapping from all coefficients to the MP parameters.

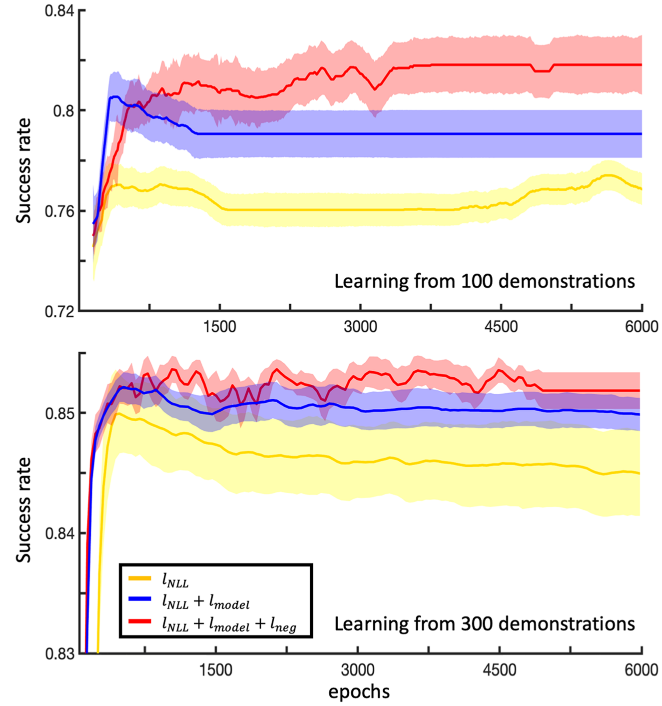

Fig. 5: Comparison between the original MDN and the entropy MDN. **Top**: the result with 100 demonstrations. **Bottom**: the result with 300 demonstrations.

In the next section, we evaluate the proposed method with four experiments. In the first two experiments, we selected the most probable mode and compared our method with previous deterministic methods. In the other two experiments, we draw multiple samples from the MDN output to either improve the performance or show the motion diversity.

IV. Experiments and Evaluations

A. Fitting Polynomials

In this experiment, we consider a 5-th order polynomial $y(x) = \sum_{k=1}^{5} a_k x^k$. The purpose is to learn a mapping from the coefficients a_k to the MP parameter $\boldsymbol{w}$ in Equation 2. The error is the distance between the true 5-th order polynomials and the generated trajectories by the output VMP parameters. We evaluate different methods separately for the inputs with one dimension a_5 to all dimensions $(a_5, a_4, ..., a_0)^T$. Figure 4 shows the results for 60 experiments. In each experiment, 30 random coefficients are for training and 20 for testing.

One-step approaches presented in [11] with SVR and GPR are denoted as ”-1” while two-steps methods are indicated with ”-2”. The symbol ”dp” for GPR refers to the dot product kernel and ”se” for the squared exponential kernel. Notice that the dot product kernel is the perfect assumption for this task because the polynomial value is indeed a dot product of the coefficients and bases. However, GPR-1 with dot product kernels is worse than other methods when learning the mapping from $(a_5, a_4)^T$ to $\boldsymbol{w}$. This result is because the time-dependent variable x is a part of the input, which loses the advantage of the correct dot product assumption. In contrast, the two-steps method GPR-2 with dot product kernels perfectly reproduces the polynomial. On the other hand, however, SVR-1 performs better than SVR-2.

For MDN, we select the most probable VMP parameter as the output. Except for the GPR with dot product kernels, MDN outperforms all other methods.

B. Random Obstacles Avoidance

To show whether the methods can scale to a more complex task than the one shown in Figure 1, we placed three obstacles randomly in the 2D space and asked for the collision-free trajectories with random starts and goals. To collect demonstrations, a person drew curves connecting random starts and goals without collisions with three randomly generated 2D balls on a tablet. Without any instructions, the human demonstrations show multiple modes and models.

The success of the task execution requires that the generated trajectory connects the start and goal without any collision with randomly placed obstacles.

Due to the task complexity and existence of the multiple modes and models, previous approaches cannot achieve acceptable results. With a 100 dataset, TP-GMM has only a success rate of about 45% with five local frames (three for the obstacles, two for the start and the goal). Both one-step and two-steps methods with SVR and GPR perform worse with a success rate that is less than 30%.

For MDN, we assume three mixture components $K = 3$. To extract the latent feature (see the blue box in Figure 3), we introduce three separate network branches for three obstacles. Each branch takes the position of one obstacle, the start and the goal of the trajectory as the input and outputs a

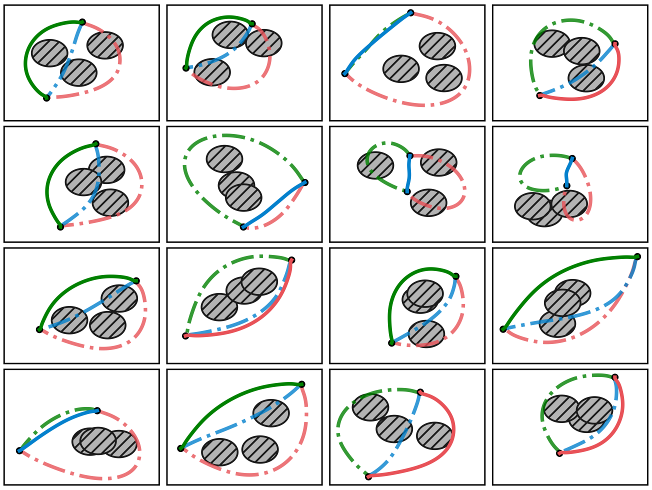

Fig. 6: Three obstacles are randomly placed in the 2D space. The **solid colorized** curves are corresponding to the most probable mode. The **dashed curves** are generated by the two other modes which are not selected by MDN.

hidden feature vector. The three hidden feature vectors are then concatenated for the rest part of the MDN. The whole MDN is trained in an end-to-end manner.

During the testing, we select the most probable mode as mentioned before. For each number of training data, 30 experiments are conducted for 30 different training datasets randomly chosen from the collected demonstrations. In order to utilize the failure cost function, after each 100 training steps, the MDN is evaluated on an evaluation dataset to produce failed samples. To avoid increasing data, we only consider the recent 3000 failed samples.

As shown in Figure 5, with 100 demonstrations, the MDN with both the entropy and failure cost functions ($l_{NLL} + l_{model} + l_{neg}$) achieves around 82% success rate. The performance is improved further to 85% with 300 demonstrations. With 100 demonstrations, the entropy MDN with the failure cost function ($l_{NLL}+l_{model}+l_{neg}$) achieves the best result and the entropy MDN ($l_{NLL}+l_{model}$) is better than the original MDN (l_{NLL}). Their difference is decreasing with the increasing number of demonstrations.

In Figure 6, 16 testing samples are shown. The colorized solid curves are generated by the most probable mode (MP parameter) given by the MDN, and the transparent dashed curves correspond to the other two modes, which are not selected by the MDN with relatively small output mixing coefficients. Three different colors indicate three different colors: The green curves bend towards the top; The red curves bend towards the bottom; The blue curves connect the start and the goal directly. As can be seen, the MDN accomplishes the task with two steps. One step is to separately update each mixture component branch to increase the success rate of their modes. The other step is to adjust the mixing coefficients output to select the mode, which has the highest chance of accomplishing the task.

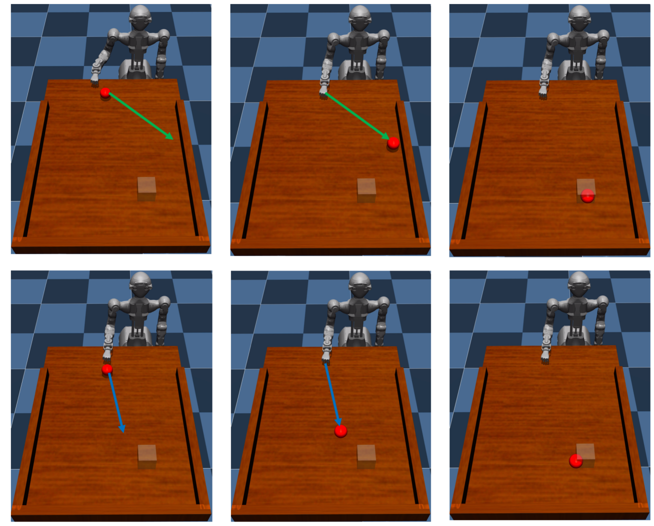

Fig. 7: In the hit ball experiment, the desired final ball location is the input of the MDN, which is denoted by a transparent box. **Top**: the robot hits the ball from its right side, and the ball bounces off the border and stops at the target. **Bottom**: the robot hits the ball directly towards the target.

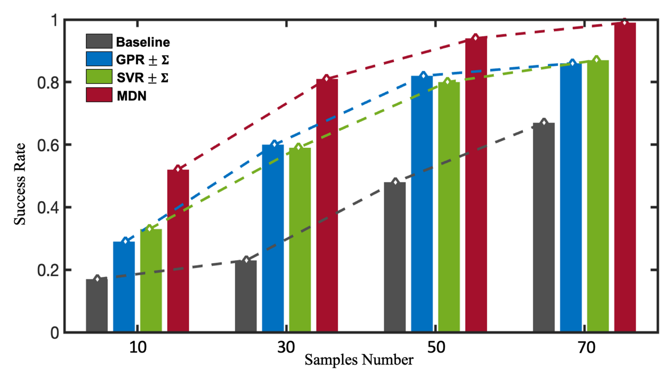

Fig. 8: The result of the hit ball experiment shows that the MDN (red) outperforms the baseline (gray), GPR$\pm\Sigma$ (blue) and SVR$\pm\Sigma$ (green)

C. *Hit Ball Experiment in MuJoCo*

In this experiment, the robot hits the ball with its fist. After being hit, the ball slides on the table and stops at some locations. The final location of the ball on the table is the task parameter query. The purpose is to generate an appropriate robot motion to hit the ball and let the ball stop at a specific location. We conducted this experiment in the MuJoCo simulator ([20]) with the model of the humanoid robot ARMAR-6 ([21]).

For the demonstrations, we use a random trajectory generator based on a 5-th order polynomial, with which the position and velocity at the end of the trajectory can be specified. The initial ball location on the table is fixed. The end velocities of the trajectories are randomly sampled from a uniform distribution. With different hitting velocities,

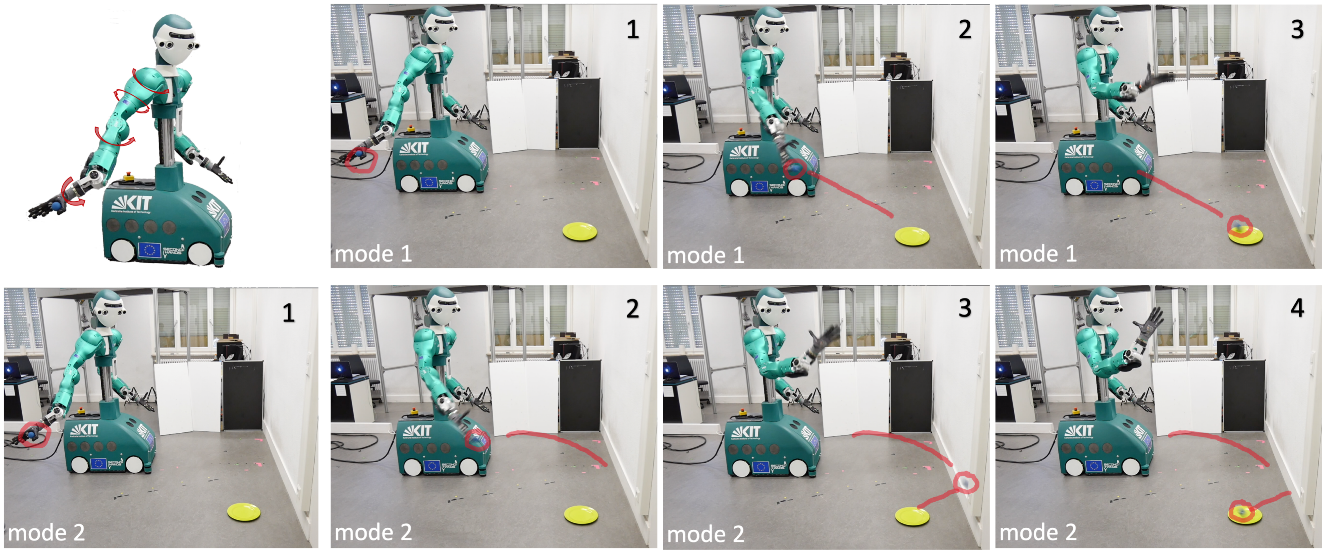

Fig. 9: **Top-left**: Four DoFs are used for throwing the ball. **Top-right**: the robot throws the ball directly to the target. **Bottom**: the robot bounce the ball to the target off the wall.

the ball stops at different locations. We collected the final locations of the ball $\boldsymbol{q}$ and the MP parameters $\boldsymbol{w}$ in a training dataset.

As shown in Figure 7, the ball can bounce off the borders of the table, which realizes multiple modes for one specific target location in the collected demonstrations. The table is $260\,\mathrm{cm} \times 200\,\mathrm{cm}$ big and the ball has radius $5\,\mathrm{cm}$. Successful task execution means that the ball is no more than $10\,\mathrm{cm}$ far from the target location.

We use $K = 3$ mixture components and train the MDN with 50 random demonstrations and test it on 100 new ball locations. If selecting the most probable mode, we get an average success rate around 15%. This poor performance might be because the task requires an accurate trajectory. With a randomly small perturbation of the MP parameters that correspond to the original 50 demonstrations, the success rate can drop rapidly. As mentioned before, to improve the task performance, we draw several samples from the output distribution. In this case, a successful MP generalization means that there exists at least one of these samples that leads to successful task execution. As shown in Figure 8, increasing the number of samples improves the success rate.

In this specific task, the number of samples helps because of two reasons. One trivial reason is the setup of the task, which allows successful task executions by chance: VMP guarantees that the robot hits the ball, and the table borders limit the final ball locations. The other reason is that the MDN learns the correct distribution, which gives a high probability to the correct MP parameter, which is unfortunately not precisely the mode. Sampling from the correct distribution has a more chance of finding the correct solution than directly selecting the most probable mode. In order to prove that the latter reason exists with MDN for this task, we consider a uniform distribution of the MP parameters as the baseline, whose interval is determined by the minimal and maximal components of the MP parameters, which correspond to the 50 demonstrations. Besides the baseline, we construct the Gaussian distributions by considering the GPR and SVR outputs as mean vectors and with a fixed variance matrix ($\Sigma = 0.01\boldsymbol{I}$).

As shown in Figure 8, MDN outperforms others for all the sample numbers. For the baseline, it coincides with the intuition that its success rate is almost proportional to the sample number because it does not learn from the demonstrations. GPR and SVR have better performances than the baseline because they draw the samples close to their output MP parameters. However, the samples drawn around their outputs are totally by chance because of the fixed variance matrix. In contrast, the MDN learns a relatively correct distribution output. Hence, it already achieves a high success rate with a smaller sample number.

D. Throw Ball Experiment

To further evaluate our methods, we let the humanoid robot ARMAR-6 throw a ball on a specific target. The arms of ARMAR-6 have eight degrees of freedom (DoF) each. In order to simplify the task, we used only four of them without loss of generality (see the four DoFs in Figure 9). The demonstrations were conducted by the human using kinesthetic teaching. After learning the corresponding MP parameter for each demonstration, we speed up the motion to 1 second and set a fixed joint goal. The robot hand always opens at 0.55 second. Then we record the location of the ball when it drops on the ground. By fixing the goals and speed of the motions, these hit ground locations are only dependent on the shapes of the joint trajectories. In the experiment, we let the robot face the wall, and the robot can bounce the ball off the wall to the target.

We let the robot throw 50 times with different human demonstrations and randomly split the collected data into 30 for training and 20 for testing. We train an MDN ($K = 2$) on 30 demonstrations. For the testing, we only use the hit ground locations of the other 20 demonstrations as task parameter queries, which guarantees that all the hit ground locations are reachable. During the testing, we place a plate on the ground to indicate the current query. Successful task execution is to throw the ball on the plate either directly or by bouncing it off the wall. In the experiment, with 10 samples, the robot missed only 2 out of 20 target hit ground locations. In Figure 9, for one specific task parameter query, we show how 2 of 10 MP parameters, which correspond to two different modes, result in different paths of the ball.

V. Conclusions and Discussions

The work addresses the problem of the movement primitive generalization to different tasks and is concerned with two aspects. First, in order to take the multiple modes and models of the human demonstrations into account, we propose to use a Mixture Density Networks (MDN) for the mapping from the task parameter query to the MP parameter distribution. The experiments show that the MDN based approach outperforms previous works. Second, to further reduce the occurrence of the mode and model collapse during training MDN, we propose the entropy cost function. Moreover, for some tasks, we introduce the failure cost to improve the performance of the MDN further. The comparison of different MDNs shows that the new cost functions perform better than the original one, especially when the set of demonstrations is relatively small.

What we do not consider here is the extrapolation of the method to areas outside the demonstrations range. Since the MDN is learned fully from demonstrations, its extrapolation capability is limited. Current methods dealing with the extrapolation problem focus only on a specific set of task parameters such as TP-GMM and via-points adaptation of VMP described in [5] or our previous work in [15]. The extrapolation of MP generalization to arbitrary task parameter queries is still unsolved.

Recent approaches such as in [13], [14] also take the task-relevant sensory inputs into account and learn them together with the robot motions. For human-robot interactions, the robot motion is generated based on the observed human activity. With the human activity considered as a task parameter query, these methods also learn a mapping from the task parameter to the MP parameter. However, unlike MDN, which directly learns this mapping, these approaches learn a generative model. In the future, we will explore the usage of our proposed method for human-robot interaction tasks and compare these methods.

For the sampling strategy, selecting the most probable mode already solves many tasks. However, for some other tasks, the suggested method needs multiple samples to achieve better performance, such as in subsection IV-C. This fact requires multiple robot trials for each task parameter query. In order to solve this problem, we consider either to use reinforcement learning to refine the mean vector given by the trained MDN with a small number of trials as in [22] or to train a discriminator, which can predict success or failure based on both task and MP parameters.

Acknowledgment

The research leading to these results has received funding from the German Federal Ministry of Education and Research (BMBF) under the project OML (01IS18040A) and the European Union Horizon 2020 Research and Innovation programme under grant agreement No 643950 (SecondHands). The authors would like to thank Jonas Rothfuss for the fruitful discussions.

References

[1] A. Billard, , S. Calinon, , R. Dillmann, , and S. Schaal, “Robot programming by demonstration,” *Springer Handbook of Robotics*, pp. 1371–1394, 2008.

[2] S. F. Giszter, F. A. Mussa-Ivaldi, and E. Bizzi, “Convergent force fields organized in the frog’s spinal cord.” *The Journal of neuroscience : the official journal of the Society for Neuroscience*, vol. 13 2, pp. 467–91, 1993.

[3] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, “Dynamical movement primitives: Learning attractor models for motor behaviors,” *Neural Comput.*, vol. 25, no. 2, pp. 328–373, Feb. 2013.

[4] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, “Probabilistic movement primitives,” in *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 2013, pp. 2616–2624.

[5] S. Calinon, T. Alizadeh, and D. G. Caldwell, “On improving the extrapolation capability of task-parameterized movement models,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov 2013, pp. 610–616.

[6] A. Ude, A. Gams, T. Asfour, and J. Morimoto, “Task-specific generalization of discrete and periodic dynamic movement primitives,” *IEEE Transactions on Robotics*, vol. 26, pp. 800–815, Oct. 2010.

[7] Y. Zhou and T. Asfour, “Task-oriented generalization of dynamic movement primitive,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 3202–3209.

[8] D. Forte, A. Gams, J. Morimoto, and A. Ude, “On-line motion synthesis and adaptation using a trajectory database,” *Robotics and Autonomous Systems*, vol. 60, p. 1327–1339, 10 2012.

[9] R. Pahic, A. Gams, A. Ude, and J. Morimoto, “Deep encoder-decoder networks for mapping raw images to dynamic movement primitives,” *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–6, 2018.

[10] B. da Silva, G. Konidaris, and A. Barto, “Learning parameterized skills,” in *Proceedings of the Twenty Ninth International Conference on Machine Learning*, June 2012.

[11] F. Stulp, G. Raiola, A. Hoarau, S. Ivaldi, and O. Sigaud, “Learning compact parameterized skills with a single regression,” in *2013 IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Oct 2013, pp. 417–422.

[12] A. Pervez and D. Lee, “Learning task-parameterized dynamic movement primitives using mixture of gmms,” *Intelligent Service Robotics*, vol. 11, no. 1, pp. 61–78, Jan 2018.

[13] E. Pignat and S. Calinon, “Learning adaptive dressing assistance from human demonstration,” *Robotics and Autonomous Systems*, vol. 93, no. C, pp. 61–75, July 2017.

[14] M. Ewerton, G. Neumann, R. Lioutikov, H. Ben Amor, J. Peters, and G. Maeda, “Learning multiple collaborative tasks with a mixture of interaction primitives,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 1535–1542.

[15] Y. Zhou, J. Gao, and T. Asfour, “Learning via-point movement primitives with inter- and extrapolation capabilities,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 4301–4308.

[16] C. M. Bishop, “Mixture density networks,” 1994.

[17] G. Mclachlan and K. Basford, “Mixture models: Inference and applications to clustering,” *Applied Statistics*, vol. 38, 01 1988.

[18] L. U. Hjorth and I. T. Nabney, "Regularisation of mixture density networks," in *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, vol. 2, Sep. 1999, pp. 521–526 vol.2.

[19] O. Makansi, E. Ilg, Ö. Çiçek, and T. Brox, "Overcoming limitations of mixture density networks: A sampling and fitting framework for multimodal future prediction," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[20] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control." in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 5026–5033.

[21] T. Asfour, L. Kaul, M. Wächter, S. Ottenhaus, P. Weiner, S. Rader, R. Grimm, Y. Zhou, M. Grotz, F. Paus, D. Shingarey, and H. Haubert, "Armar-6: A collaborative humanoid robot for industrial environments," in *2018 IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, 2018, pp. 447–454.

[22] J. Kober and J. Peters, "Reinforcement learning in robotics: A survey," *Learning Motor Skills: From Algorithms to Robot Experiments*, pp. 9–67, 2014.