2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 24–28, 2017, Vancouver, BC, Canada

Task-Oriented Generalization of Dynamic Movement Primitive

You Zhou and Tamim Asfour

***Abstract*— An important question in imitation learning is how to generalize a learned motion to novel situations. The motion generalization depends on a set of features which can be represented as feature vectors spanning a feature space, called query space. The purpose of generalization is to find a mapping from this query space to the motion primitive space (MP space). In this paper, we address the problem of generalization of dynamic movement primitives (DMPs) to new queries by applying locally weighted regression (LWR) with radial basis functions (RBF). Since two DMPs differ only in their non-linear part, we transform the problem of DMP generalization to a regression analysis problem. We introduce a task-oriented regression algorithm with a cost function that takes task constraints into consideration and which relies on model switching to solve the problem of poor DMP generalization when using a single regression model for the entire query space. The evaluation shows that our algorithm outperforms related approaches in the literature in terms of generalization capabilities.**

I. Introduction

The generalization of a motion primitive to different and novel situations is not trivial. The world in which a robot is acting can be described or abstracted by parameters, which construct a space which we call *query space*. It is always difficult to find a general mapping from the query space to the MP space, because each query can be corresponding to several MPs. Unfortunately, there are infinite number of MP solutions for one query, but not many of them are of good quality. One category of the solutions to this problem, including GMM [1] and ProMP [2], is to generate a movement according to a trajectory distribution, which might capture a lot of variations in the demonstrations. The downside of these stochastic movement representations is that it is difficult to introduce the dynamic change to them during the execution. On the other hand, a dynamic system such as a Dynamic Movement Primitive [3] can easily handle dynamic change in the environment by appending a coupling term to the system. Although it is not obvious how to capture the variations by DMP, it can be done by training one DMP for each demonstration and conducting principle component analysis (PCA) on the DMP weight vectors [4]. The DMP representation is a good choice for MP generalization because each DMP can be described by a single weight vector.

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement no 643666 (I-Support) and the DFG Priority Program 1527 ”Autonomous Learning” funded by the German Research Foundation (DFG).

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

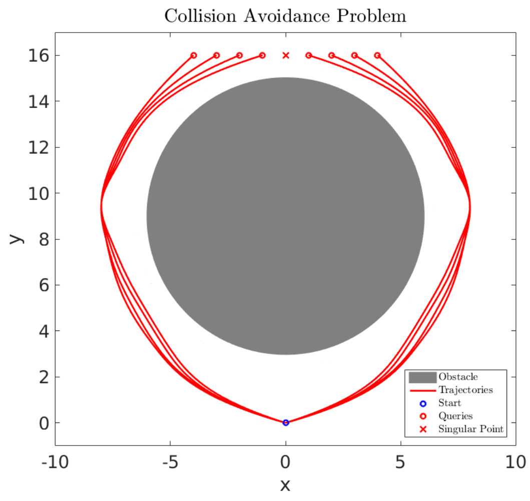

Fig. 1: This obstacle avoidance problem is used in the paper to compare our method with the method introduced in the previous work [5]. The gray circle represents the obstacle. All our trajectories start from the blue round point and end at the red round points, which are considered as query points. For each query point, there is a corresponding red demonstrated trajectory. The red cross indicates the singular point. The task is to firstly learn these demonstrations by imitation learning approaches and then generalize the learned motion to new query points. The previous method in [5] will fail when the query is close to the singular point.

We follow a basic assumption that there exists a mapping from the query space to DMP space, which is a vector space containing DMP weight vectors. Moreover, we assume that this mapping is learnable by observing a training dataset. After successful learning of the mapping, we can use it to find an appropriate DMP for every new query out of the training dataset.

The reinforcement learning is an another learning strategy for DMP weight vectors other from imitating a demonstrator [6]. It introduces a task-oriented reward or cost function to maximize or minimize, thus, guarantees that the learned DMP can generate a proper trajectory to accomplish the task. If the task-oriented function is easily formulated, the reinforcement learning can improve the quality of the training dataset, hence, makes the DMP generalization more suitable or accurate for the task.

In the next section, we introduce previous work regarding the DMP generalization. After that, we introduce and compare our method with the previous work.

II. Related Works

In [5], the authors assumed that the similarity in the query space should imply the similarity in the DMP space. In order to find an appropriate DMP for a new query, the contribution of each learned DMP is determined by the distance between its corresponding query and the new query in the query space. The method described in [5] has several constraints. The one is that the training query must be regularly distributed in the query space. If this was not the case, the generated DMP for a new query would be similar to those training DMPs with queries in the high density area. An another constraint is that the new query must be away from the boundary of the training queries set, otherwise, the suggested regression algorithm cannot find an expected DMP because of the boundary problem.

Furthermore, the query similarity cannot simply imply the similarity of DMPs, hence, the similarity of trajectories. Because a query space with a limited dimension is always insufficient to represent the task constraints accurately, which are important for DMP generalization. For example, in Fig 1), we have a collision avoidance problem, where the trajectories always end up with the same y-axis value. One intuitive design of a query space is to include all possible x-axis values of the goals. In this case, the similarity of the queries which are close to the singular point shown as a red cross cannot result in the similarity of DMPs learned by imitation learning for these red demonstrated trajectories. In this case, the basic assumption mentioned in [5] is not correct, thus, it cannot give a proper DMP solution.

To solve this problem, we replace kernel regression (KR), which is used in [5] with locally weighted regression (LWR) and compare the results in the context of the collision avoidance problem (see Sec. III-C). Considering that the reinforcement learning can improve the quality of the training dataset, we introduce a task-oriented loss function into the regression cost function, which is minimized during learning the regression model (see Sec. III-E). After that, we modify the model switching algorithm (see Sec. III-F) introduced in [7] to address the problem where the similarity criterion is not suitable. Finally, we give a complete algorithm (see Sec. IV) and a real application (see Sec. V)) to demonstrate the benefit of our method.

III. Task-Oriented DMP Generalization

A. Dynamic Movement Primitive

Dynamic Movement Primitive (see eq 1) consists of two parts, namely a transformation system, which is a non-linear dynamic system created by adding a non-linear force term $f(.)$ to a damped spring system, and a canonical system describing the dynamic of the canonical value x which makes the transformation system time-independent.

$$\begin{aligned} \tau\ddot{y} &= K(g-y) - D\dot{y} + f(x) \\ \tau\dot{x} &= -\alpha x, \end{aligned} \tag{1}$$

where y represents the DMP state. K and D are stiffness and damper factors for the damped spring system. The first equation describes the transformation system, while the second one is a simple canonical system. The force term $f(x)$ is a function of x, which can be learned by using LWR such that

$$f(x) = \frac{\sum_{i=1}^{n} \phi_i(x) w_i}{\sum_{i=1}^{n} \phi_i(x)}, \tag{2}$$

where phi is the kernel function and $\boldsymbol{w}$ is the weight vector, which is unique for each DMP. If the hyper parameters of the DMP, such as goal g and temporal factor τ, are fixed and the canonical system is determined, one weight vector $\boldsymbol{w}$ determines one trajectory. Hence, we have a mapping from the DMP space to the trajectory space.

B. Kernel Regression

Locally weighted regression is a useful regression technique to approximate a target function. In [5], the authors used LWR, but with a constant function model. They assumed that the desired mapping is locally constant, which is rarely to be true. In literature, the approach of fitting constants using a locally weighted training criterion is known as kernel regression [8].

In order to give a clear description of our approach, we provide firstly a detailed introduction to the work in [5], where the authors suggested to minimize the following objective function:

$$\sum_{k=1}^{M} \|\boldsymbol{X}_k \boldsymbol{w} - \boldsymbol{f}_k\|^2 K(d(\boldsymbol{q}, \boldsymbol{q}_k)), \tag{3}$$

where M is the number of training samples, $\boldsymbol{f}_k$ can be considered as k^{th} DMP because the force term itself characterizes DMP. $d(\boldsymbol{q}, \boldsymbol{q}_k)$ is the distance measure in the query space. $\boldsymbol{X}_k$ is the corresponding basis function values matrix w.r.t the canonical values when executing the trajectory, and we can consider it unchanged for different training trajectories, because it is easy to normalize all trajectories to have the same timestamps, hence same canonical values (see details in [5]). K is the distance kernel function defined as following:

$$K(d) = \begin{cases} (1-|d|^3)^3, & \text{if } |d| < 1 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

which is called tricube kernel function. After calculating derivative of the objective function in equation 3 with respect to $\boldsymbol{w}$, we get the zero point that

$$\boldsymbol{w}(\boldsymbol{q}) = \frac{\sum_{k=1}^{M} K(d(\boldsymbol{q}, \boldsymbol{q}_k)) \boldsymbol{w}_k}{\sum_{k=1}^{M} K(d(\boldsymbol{q}, \boldsymbol{q}_k))}, \tag{5}$$

where $\boldsymbol{w}_k$ is the weight vector for k^{th} DMP in the training samples. This is a simple weighted average method, which is conducted in the original query space. The kernel function K can represent user's preference. By using tricube kernel function, all samples near to the query in some senses $(d < 1)$

are considered to have similar effect on the result while other samples have no impact, because the tricube kernel function has a steep descent at the boundary where $d = 1$.

For data distributed on a regular grid away from any boundary, kernel regression and locally weighted regression with non-constant local models are equivalent. However, for irregular data distributions there is a significant difference, and LWR has many advantages over kernel regression, see [8] [9] [10].

C. Locally Weighted Regression

Let us assume that we have a mapping from query space to DMP space to be learned. Each DMP can then be represented by a real weight vector $\boldsymbol{w} \in \mathbb{R}^N$. Now we assume that the mapping should look like the following:

$$w_j(\boldsymbol{q}) = \frac{\sum_{i=1}^{N} \psi_i(\boldsymbol{q})\eta_{ji}}{\sum_{i=1}^{N} \psi_i(\boldsymbol{q})}, \tag{6}$$

where $w_j(\boldsymbol{q})$ is the j^{th} component of the DMP weight vector and it is a function of $\boldsymbol{q}$. $\boldsymbol{\eta_j}$ is the weight vector to be learned and ψ is a kernel function for locally weighted regression. We can rewrite the above formulation as following by including all weight vector components:

$$\boldsymbol{w}(\boldsymbol{q}) = \frac{1}{Z}\boldsymbol{H}^T\boldsymbol{\Psi}(\boldsymbol{q}), \tag{7}$$

where $Z = \sum_{i=1}^{N} \psi_i(\boldsymbol{q})$, and

$$\boldsymbol{\Psi}(\boldsymbol{q}) = \begin{pmatrix} \psi_1(\boldsymbol{q}) \\ \cdots \\ \psi_N(\boldsymbol{q}) \end{pmatrix}, \quad \boldsymbol{H} = \begin{pmatrix} \eta_{11} & \cdots & \eta_{N1} \\ \vdots & \ddots & \vdots \\ \eta_{12} & \cdots & \eta_{NN} \end{pmatrix},$$

where the first column of the matrix $\boldsymbol{H}$ is related to the first component of DMP. For the j^{th} component of the weight vector, we minimize the squared error:

$$L_j = \sum_{k=1}^{M}\sum_{i=1}^{N} \psi_i(\boldsymbol{q}_k)(\eta_{ji} - w_j(\boldsymbol{q}_k))^2, \tag{8}$$

where M is the number of samples and $\{(\boldsymbol{q}_k, \boldsymbol{w}_j(\boldsymbol{q}_k))\}_{k=1:M}$ is a set of training samples. After calculating the derivative and its zero point, we get the optimized weight component η_{ji}:

$$\eta_{ji} = \frac{\sum_{k=1}^{M} \psi_i(\boldsymbol{q}_k)w_{jk}}{\sum_{k=1}^{M} \psi_i(\boldsymbol{q}_k)}, \tag{9}$$

where $w_{jk} = w_j(\boldsymbol{q}_k)$. We replace η_{ji} in 6 and get

$$w_j(\boldsymbol{q}) = \frac{\sum_{k=1}^{M}\sum_{i=1}^{N} \psi_i(\boldsymbol{q}_k)\psi_i(\boldsymbol{q})w_{jk}}{\sum_{k=1}^{M}\sum_{i=1}^{N} \psi_i(\boldsymbol{q}_k)\psi_i(\boldsymbol{q})}. \tag{10}$$

We observe that the result is actually similar to the kernel regression and we can use the vector product notation and get

$$w(\boldsymbol{q}) = \frac{\sum_{k=1}^{M} k(\boldsymbol{q}_k, \boldsymbol{q})w_k}{\sum_{k=1}^{M} k(\boldsymbol{q}_k, \boldsymbol{q})}, \tag{11}$$

where

$$k(\boldsymbol{q}_k, \boldsymbol{q}) = \langle \boldsymbol{\Psi}(\boldsymbol{q}_k), \boldsymbol{\Psi}(\boldsymbol{q})\rangle\,. \tag{12}$$

The function k is a similarity measure in a high dimensional space. $\boldsymbol{\Psi}$ is the mapping which we are looking for, which maps the query space on a high dimensional space. The parameters of the mapping $\boldsymbol{\Psi}$ decides the accuracy of the mapping from query space to weight vector space. The difference between KR and LWR is that k in KR is a scalar valued function depending only on the distance d while it is a vector-valued function in LWR, which is considered as a measure in a higher dimensional space.

D. Parameters of Kernel Function

One important kind of kernels used for locally weighted regression are radial basis functions (RBF) as given in equation 13.

$$\psi_i(\boldsymbol{q}) = -\frac{1}{2}exp(-(\boldsymbol{q} - \boldsymbol{c_i})^T\Sigma_i(\boldsymbol{q} - \boldsymbol{c_i})), \tag{13}$$

where $\boldsymbol{c_i}$ is the center of the i^{th} kernel and $\boldsymbol{\Sigma_i}$ decides the spreads of the kernel in each dimension.

In order to get suitable parameters for kernel function, we leave one sample out and calculate the target weight for this sample using equation 11 as following:

$$\hat{\boldsymbol{w}}_i(\boldsymbol{\theta}) = \frac{\sum_{k=1,k\neq i}^{M} k_{\boldsymbol{\theta}}(\boldsymbol{q}_k, \boldsymbol{q}_i)\boldsymbol{w}_k}{\sum_{k=1,k\neq i}^{M} k_{\boldsymbol{\theta}}(\boldsymbol{q}_k, \boldsymbol{q}_i)}, \tag{14}$$

where

$$\boldsymbol{\theta} = (\{\boldsymbol{c_i}, \boldsymbol{\Sigma_i}\}_{i=1}^{N}, N), \tag{15}$$

and $\hat{\boldsymbol{w}}_i$ is the i^{th} DMP in the training dataset. The parameters are now obtained by minimizing the errors between the DMPs:

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{M} ||\hat{\boldsymbol{w}}_i(\boldsymbol{\theta}) - \boldsymbol{w}_i||^2. \tag{16}$$

Furthermore, it is not difficult to see the equivalence between the above equation and the following one:

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{M} dist(\hat{\boldsymbol{T}}_i, \boldsymbol{T}_i), \tag{17}$$

where $\boldsymbol{T}_i$ is the trajectory generated by i^{th} training DMP and $\hat{\boldsymbol{T}}_i$ is the trajectory generated by the DMP given by the regression model. $dist(*, *)$ is a distance function in the

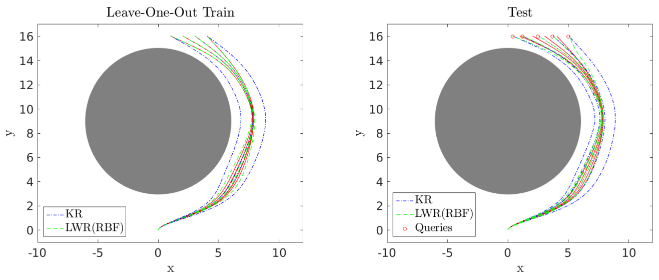

Fig. 2: On the left, the two methods (KR and LWR with the number $N = 3$ of kernels) are trained on the whole training dataset except one training query point, then a trajectory is generated for this query point. On the right, the trajectories generated by two methods for the test data $\{0.4, 1.2, 2.5, 3.7, 5.0\}$. The nearer the query point is to the boundary, the worse the performance of the KR. In contrast, LWR has always a good result.

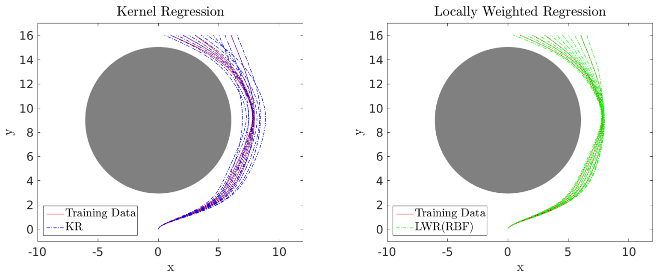

Fig. 3: The two methods are used for irregularly distributed training dataset. The left figure is for simple kernel regression, while the right one is for suggested locally weighted regression. The training dataset are the red lines and the results are blue dotted or green dotted lines. The result shows that LWR outperforms KR with all trajectories in a more narrow path. We can also observe that the irregularly distributed training dataset has a great impact on the KR results.

trajectory space such as DTW (Dynamic Time Wrapping). Note that we fix start, goal position and temporal factor to make sure that one DMP decides one trajectory.

This LWR is actually a normalized RBF network, whose parameters are learned by gradient descent. The weight $\boldsymbol{\eta}$ corresponds to the linear output layer of RBF network and is easily learned by least squares when the hyper parameters of the kernel functions are fixed. A large number of the basis functions N might cause the over-fitting and the high computational cost. In the training phase, we make sure that $N < M$, otherwise the above method can always guarantee that the fitting curve will go through all the training samples(over-fitting). In Fig 2 and Fig3, RBF based LWR has better performances than KR in the collision avoidance. Hence, our method uses RBF-based LWR in order to address the regression problem raised by irregularly distributed training datasets or the queries near to the boundary.

E. *Task-Oriented Generalization*

With a good regression model, we can generate a theoretically good trajectory for a new query point. However, the robot's execution in a real world is not as simple as a trajectory planner. The robot has to be able to react to unexpected events and situations, in which a safe trajectory is needed instead of a trajectory similar to a demonstration. Moreover, imitating a trajectory using DMP is not error-free, especially when the regression algorithm for approximating the force term of the DMP is not good enough to compensate the error between the demonstrated and the generalized trajectory. This occurs when using LWR with an insufficient number of kernel functions. To address these issues it is necessary to consider the costs of a trajectory associated with the task.

In order to achieve a task-oriented generalization, we define a more suitable loss function by adding the task-oriented loss to the original cost function and get equation 18.

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{M} \left(\alpha||\hat{\boldsymbol{w}}_i(\boldsymbol{\theta}) - \boldsymbol{w}_i||^2 + \beta R(\hat{\boldsymbol{w}}_i(\boldsymbol{\theta}))\right) \quad (18)$$

The first part of equation 18 is the cost for regression analysis and the second part is the task-oriented cost $R(\hat{\boldsymbol{w}}_i(\boldsymbol{\theta}))$. These two parts of the cost functions might have conflicts with each other. To address this problem, we can introduce the weights α and β, with $\alpha = 1 - \beta$ to describe the interdependency between the two parts. This total cost function not only guarantees that the regression model can reproduce DMP for a training query point when the imitation learning with DMP is almost error-free and the human demonstration is also suitable for the task, but also ensures that the task will be fulfilled (when β is relatively large.) even if the imitation learning has big errors or the human demonstration itself is inappropriate.

It is not easy to directly solve the optimization problem given in equation 18, especially when $R(\hat{\boldsymbol{w}}_i(\boldsymbol{\theta}))$ is an integral of the immediate cost function r_t over the time duration $t \in [0, T]$. This optimization problem has the similar form with the one which the reinforcement learning algorithms try to solve. One recently developed reinforcement learning algorithm for direct policy improvement is PI^2 algorithm [11]. However, we cannot use PI^2 directly for the cost function in equation 18 because $\boldsymbol{\theta}$ is not a set of policy parameters but parameters of a regression model. It is possible to consider $\boldsymbol{\theta}$ as parameters of the policy including the choice of the regression model, but it is very difficult to derive an appropriate algorithm. One simpler solution is to adjust the DMP weights for the training dataset before performing regression. In this case, we change the regulator in the immediate cost function as following:

$$r_t = \beta c_t + \alpha(\boldsymbol{w}_t - \boldsymbol{w})^T \boldsymbol{Q}(\boldsymbol{w}_t - \boldsymbol{w}), \quad (19)$$

where c_t is the immediate cost and $(\boldsymbol{w}_t - \boldsymbol{w})^T \boldsymbol{Q}(\boldsymbol{w}_t - \boldsymbol{w})$ is the regulator to ensure that the resulting DMP weight vector is not far away from the weight vector in the training dataset,

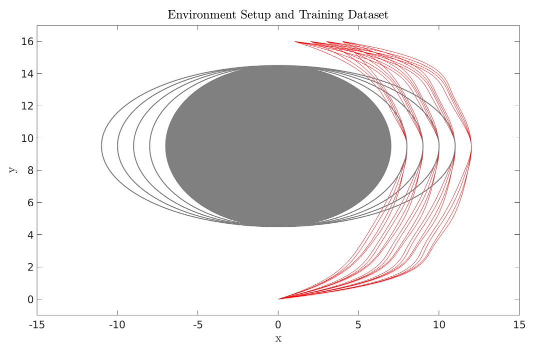

Fig. 4: The gray ellipses are the obstacles with different horizontal radii, which are {7,8,9,10,11} from small to big. The training dataset contains trajectories for different target positions and obstacle radii, thus, the query space is a 2D space. In the training phase, we choose randomly a query point and alter the corresponding trajectory to make it collide with the obstacle as an outlier of training samples. The regression result is shown in Fig. 5.

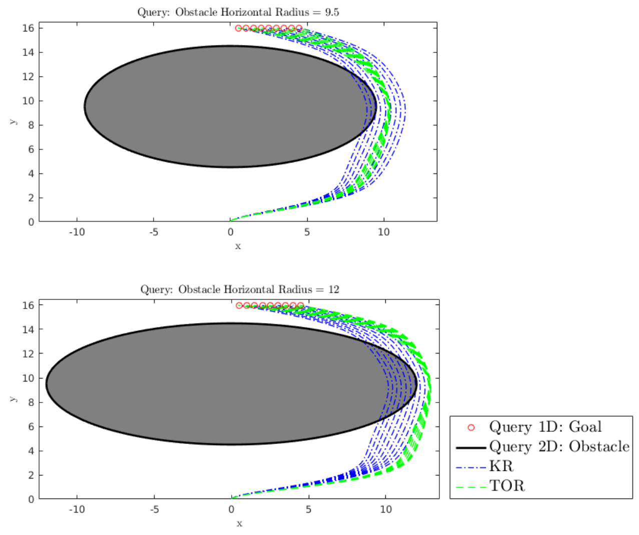

Fig. 5: The original KR method without considering the possible collisions cannot generate collision-free trajectories for some queries, while our method with a number $N = 10$ of kernels can always generate a suitable trajectory.

which is given by the imitation learning. Then we train the regression model based on the new dataset $\{(\boldsymbol{q}_k, \boldsymbol{w}_k^{rl})\}_{k=1:M}$ with

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}\in\Theta} \sum_{k=1}^{M} ||\hat{\boldsymbol{w}}_k(\boldsymbol{\theta}) - \boldsymbol{w}_k^{rl}||^2. \tag{20}$$

In order to clarify the benefit of this kind of cost functions, let us consider a more complex example than the simple collision avoidance example introduced previously. We consider a 2D query space where the 1^{st} dimension represents the x-axis value of the goal and the 2^{nd} dimension represents the horizontal size of the ellipse obstacle(see Fig 4).

By using the equation 18 as the regression cost function, we obtain a task-oriented model, whose responses nearly hit the obstacle for queries in or near to the range of training queries, when β is relatively large. However, it is possible that a certain regression model with any parameter cannot avoid collisions for all queries in a training dataset. In this case, a model switching (see Sec. III-F) should be undertaken, which means that we need to split the training dataset into different parts.

F. Model Switching

Testing almost all parameters of the regression model would not necessarily result in reduction of the cost function value below a certain threshold. Hence, we need to split the training samples and create different models for different subsets of training dataset. This technique is called model switching. we use a slightly modified method based on [7] to realize model switching. We consider the posterior probability:

$$p(m|\boldsymbol{q}, \boldsymbol{w}) \propto p(\boldsymbol{w}|m, \boldsymbol{q}) \cdot p(m|\boldsymbol{q}), \tag{21}$$

where m is the model that we are looking for, $\boldsymbol{q}$ is the query point, and $\boldsymbol{w}$ is the weight vector in the original training dataset.

$$p(\boldsymbol{w}|m, \boldsymbol{q}) = \frac{1}{\sqrt{2\pi}\sigma} \cdot exp\left\{-\frac{c(\boldsymbol{q}, \boldsymbol{\theta}_m)}{2\sigma}\right\}, \tag{22}$$

where

$$c(\boldsymbol{q}, \boldsymbol{\theta}_m) = \alpha||\boldsymbol{w} - \boldsymbol{w}(\boldsymbol{q}, \boldsymbol{\theta}_m)||^2 + \beta R(\boldsymbol{w}(\boldsymbol{q}, \boldsymbol{\theta}_m)) \tag{23}$$

and $w(\boldsymbol{q}, \boldsymbol{\theta}_m)$ is the result given by the m^{th} regression model with the parameters set $\boldsymbol{\theta}_m$. σ defines the spread of Gaussian distribution. We set $\sigma = 0.5$. Unlike in [7], we calculate the conditional probability by considering the whole cost function defined in the equation 18 and making the model switching task-oriented.

$p(m|\boldsymbol{q})$ is the prior probability of models, which has a local support to avoid the interference from the query points that are not in the neighbourhood of the model center point $\boldsymbol{c}_m$. The neighbourhood is defined according to [7] such that

$$p(m|\boldsymbol{q}) = 0 \iff \exists j, \langle \boldsymbol{q} - \boldsymbol{c}_j, \boldsymbol{c}_m - \boldsymbol{c}_j \rangle < 0, \tag{24}$$

where $\langle\cdot,\cdot\rangle$ is the scalar product. Instead of attempting to create a smooth bridge between two models, we consider the models as independent of each other, which is a more intuitive assumption. For example, in Fig. 6, the nearer the query point is to the singular query $q = 0$ from the positive side, the larger the first weight component should be to ensure a large positive force at the beginning. In contrast, the first weight component needs to be negative, when the query point goes to zero from the negative side.

In the training phase, an Expectation Maximization (EM) algorithm is used like in [7]. In the E-phase, we calculate the posterior probability $p(m|\boldsymbol{w}, \boldsymbol{q})$ and group the data according to the maximal one. In the M-phase, we estimate each model's parameters by minimizing the cost function 18.

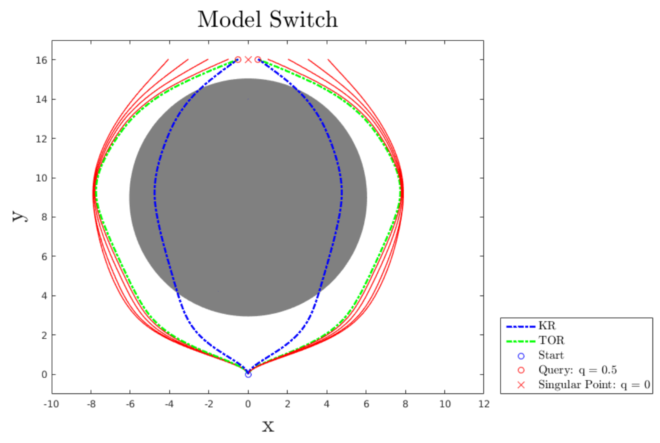

Fig. 6: The original method fails to generate a trajectory without collision for the queries near to the singular point $q = 0$, while our method has no problem with these queries.

In Fig. 6, we show the solution to the obstacle avoidance problem where the query space is limited and it is necessary to split the query space into subspaces to generate multiple regression models. The results show that LWR with model switching can handle all queries, while the original kernel regression without model switching fails to provide an appropriate solution, especially for the queries near to the singular point $q = 0$.

IV. Task-Oriented Regression for DMP Generalization

All the above discussions end up with the algorithm 1. The relationship between α and β depends on the user's preference about which cost function part plays a more important role. For example, in our obstacle avoidance task, the immediate cost c_t should be taken seriously, thus, $\alpha < \beta$. In other examples, c_t might be only related to the acceleration and the imitation learning is the most important purpose, thus, $\alpha > \beta$. P_{min}, namely the minimum of the acceptable posterior probability, determines how easy a model is accepted by a query point. If P_{min} is large, the algorithm only keeps the query points with high probabilities for a certain model. NUM_{min} determines how easy a new model is constructed. If NUM_{min} is large, the algorithm tends to consider the queries with relatively low probabilities for all models as outliers instead of creating a new model. Both P_{min} and NUM_{min} are determined by the user's confidence of the correctness of the training dataset. If the training dataset contains a lot of outliers, both values should be large, otherwise, they should be small. Once the number of outliers is larger than NUM_{min}, then a model is constructed and removed repeatedly and the algorithm will never converge because no model can be learned based on outliers. If a model is constructed and removed repeatedly, either we need to increase NUM_{min} or we throw away the queries with the lowest probability instead of keeping them in the unsorted set D until P_{min} is satisfied for all the rest queries or the size of the unsorted set D is smaller than NUM_{min}.

```
Algorithm 1: Task-Oriented Regression Algorithm
Input: $\{(\boldsymbol{q}_i, \boldsymbol{w}_i)\}_{i=1}^{M}$: training dataset
       $P_{min}$: the minimum probability threshold
       $NUM_{min}$: the minimum size of the models
Output: $\{\boldsymbol{\theta}_m\}_{m=1}^{K}$ where $K$ is the final number of the
        regression models.
Initialize $\boldsymbol{\theta}_{m_0}$ corresponding to $m_0$;
Initialize $\eta(i, m_0) = 1$ and the model set $\Xi = \{m_0\}$;
while model switched or created do
    Step1: Reinforcement Learning:
    Get $\boldsymbol{w}^{rl}$ with the immediate cost function:
        $r_t = \alpha(\boldsymbol{w}_t - \boldsymbol{w})^T \boldsymbol{Q}(\boldsymbol{w}_t - \boldsymbol{w}) + \beta c_t;$    (25)
    Create the dataset $\{(\boldsymbol{q}_i, \boldsymbol{w}_i^{rl})\}_{i=1}^{M}$;
    Step2: Model Switching:
    for each data point $(\boldsymbol{q}_j, \boldsymbol{w}_j)$ do
        M-step:
        Learn $\boldsymbol{\theta}_m$s with the regression cost function:
            $\boldsymbol{\theta}_m = \arg\min_{\boldsymbol{\theta}\in\Theta} \sum_{i=1, i\neq j}^{M} \eta(i,m) ||\hat{\boldsymbol{w}}_i(\boldsymbol{\theta}) - \boldsymbol{w}_i^{rl}||^2.$    (26)
        E-step:
        Calculate $p(m|\boldsymbol{q}_j, \boldsymbol{w}_j)$ for each model $m \in \Xi$;
        if $\exists m \in \Xi, p(m|\boldsymbol{q}_j, \boldsymbol{w}_j) \geq P_{min}$ then
            Calculate $\eta(j, m)$:
                $\eta(j,m) = \begin{cases} 1 & m = \arg\max_{m\in\Xi} p(m|\boldsymbol{w}_j, \boldsymbol{q}_j) \\ 0 & otherwise \end{cases}$    (27)
        else
            Put data $(\boldsymbol{q}_j, \boldsymbol{w}_j)$ into an unsorted dataset
            $D$;
        end
    end
    if $|D| \geq NUM_{min}$ then
        Create a new model $m$ and put it in $\Xi$;
        Clear $D$ ;
    else
        Count these data points as outliers;
    end
    Remove the model with zero associated data points.
end
```

If PI^2 is used for reinforcement learning, this part can be moved out of the while-loop. After reinforcement learning gives us a set of adjusted training dataset, we learn a regression model on this new dataset. However, in reality, the PI^2 algorithm changes each component of the weight vector in a similar way, which does not guarantees that the newly generated DMP trajectory has a desired shape similar to the shape given by the initial DMP. Moreover, PI^2 is only a benefit when we are not sure about the direction in which the DMP weight vector should be altered. If we assume that most of the training samples in each set for a model are obtained

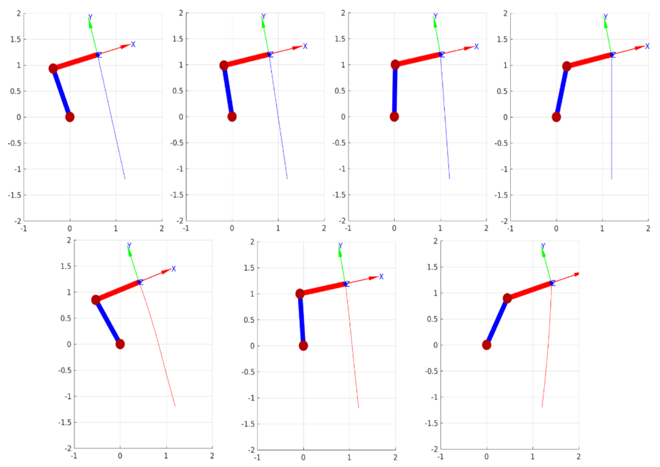

Fig. 7: The approaching problem with a simulated two joints 2D robot is shown here. Each of the training joint trajectories shown in the first row creates a straight line in the task space. We consider a simple 1D query space for the x-axis value of the goal. The training queries from left to right are $\{0.6, 0.8, 1.0, 1.2\}$. The second row shows the results for three representative query points. They are from left to right $\{0.4, 0.9, 1.4\}$. The joints trajectories are shown in Fig. 8.

by an imitation learning of a good quality, we can use the current regression model to get an expected DMP weight vector $\hat{\boldsymbol{w}}$ and the direction to change the current weight $\boldsymbol{w}$ is now $\hat{\boldsymbol{w}} - \boldsymbol{w}$. We need to learn the rate λ and update $\boldsymbol{w} = \boldsymbol{w} + \lambda(\hat{\boldsymbol{w}} - \boldsymbol{w})$ to minimize the integral cost $\int_{t=0}^{T} r_t$. In this case, the reinforcement learning part should be in the while-loop, because $\hat{\boldsymbol{w}}$ is given by the current models. Furthermore, it is also possible that $\boldsymbol{w}$ is quite different from $\hat{\boldsymbol{w}}$ and they even generate different shapes of trajectories. We can alter the immediate cost in equation 25 to have an extra term for $\hat{\boldsymbol{w}}$.

$$\begin{aligned} r_t = &\alpha(\boldsymbol{w}_t - \boldsymbol{w})^T \boldsymbol{Q}_1 (\boldsymbol{w}_t - \boldsymbol{w}) \\ &+ \alpha(\boldsymbol{w}_t - \hat{\boldsymbol{w}})^T \boldsymbol{Q}_2 (\boldsymbol{w}_t - \hat{\boldsymbol{w}}) \\ &+ \beta c_t. \end{aligned} \tag{28}$$

V. Application

The task-oriented generalization of DMP has a wide variety of applications. Here, we illustrate one application which enables a DMP trained in the joint space to be generalized for a set of given task space constraints. A task space DMP, namely a DMP trained in task space, can easily adapt to a new goal or to a new start position defined in the task space. However, it is not trivial to adapt a joint space DMP to a task space goal. We can represent all task space goals as queries. By doing this, we transform the problem to finding a mapping from query space to DMP space, which can be solved by the techniques we described above.

In Fig 7 and Fig 8, a two joints robot accomplishes a point-to-point approaching with different goals.

Any task space constraints can be satisfied by generalization of a joint space DMP, if we consider them as queries.

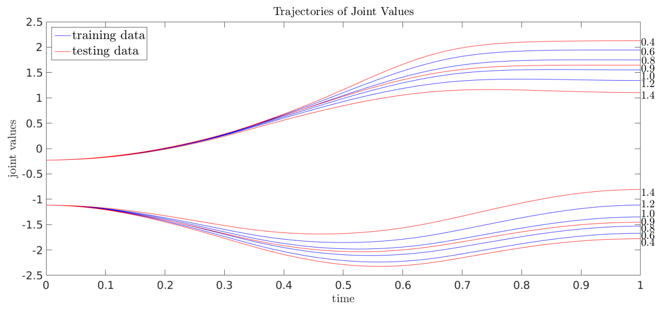

Fig. 8: The blue trajectories are for the training queries while the red ones for the testing queries. The numbers listed on the right side of the diagram indicate which trajectories are for which queries.

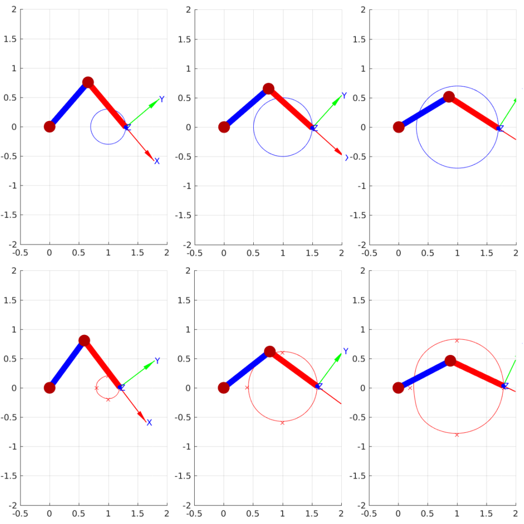

Fig. 9: The two joints arm is learning how to draw a circle with different sizes. Each of the training data shown in the first row is corresponding to a circle with a different size in the task space. We consider that the radii of the circles construct a query space. The training queries from left to right are $\{0.3, 0.5, 0.7\}$. The second row shows the results for three queries $\{0.2, 0.6, 0.8\}$. The joints' trajectories are shown in Fig. 10.

In Fig 9 and Fig 10, we show that the arm draws circles in a 2D space. After three demonstrations, the arm can adapt to different sizes of the circles.

An other experiment is also shown in the attached video, where a 2D grasping task is solved by generalization of DMPs with our generalization algorithm. The query space is defined by the object size (radius) and its 2D pose. DMPs are learned in the joint space and a regression model is learned to generate new DMPs for new queries.

More experiments (see video) are conducted in the simulator where the model of our humanoid robot ArmarIII is available [12] [13] [14].

VI. Conclusions and Future Works

In this work, we presented an algorithm to learn the mapping from query space to DMP space, which considers

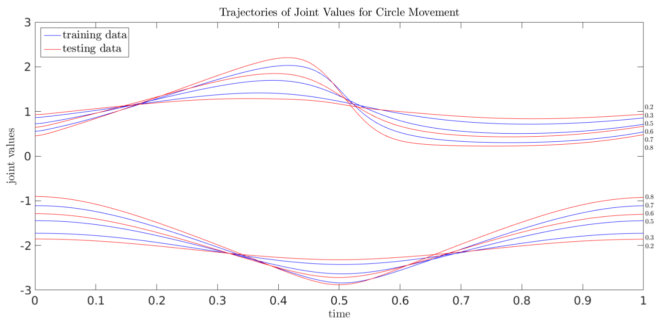

Fig. 10: The blue trajectories are for the training queries while the red ones for the testing queries. The numbers listed on the right side of the diagram indicate which trajectories are for which query points.

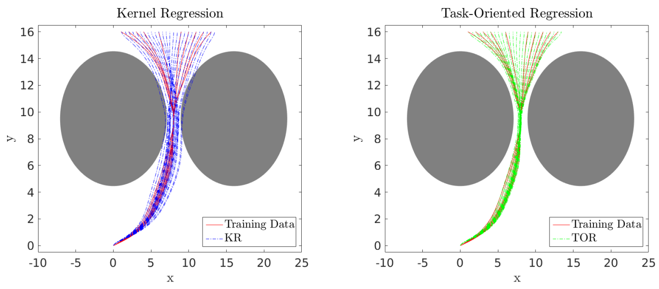

Fig. 11: The query space is a one-dimensional space about the goal of the trajectory. The blue dotted lines are given by KR method, which cover a wide range and collide with the obstacles for some query points, while the green dotted lines generated by our suggested regression method show almost the via-point behaviour and none of them have collision with the obstacles.

not only errors in imitating learned motions, but also the task-specific costs. The resulting mapping generates more meaningful DMP weight vectors for different query points than the one given by the work [5]. We showed that the algorithm outperforms similar previous work.

The fascinating thing regarding the task-oriented DMP generalization is that we can treat anything as queries without considering its physical or mathematical relationship with the task. This relationship is actually learned by some regression functions. For example, in the original case, if we want to drive a robot to capture a cup on the table, we need to firstly segment the cup from the background in the image space and then transform the cup's position from camera frame to robot's frame. After that, we use inverse kinematics to get a set of joints' trajectories. With query based DMP generalization, however, we can consider the pixels representing the cup in the image space as queries and get directly the motion in the joint space instead of using geometric transformation and inverse kinematics.

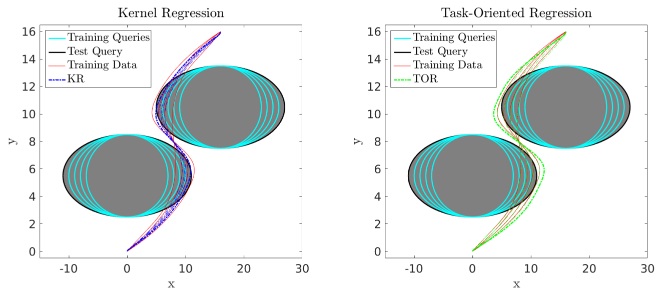

Fig. 12: The query space is a one-dimensional space about the horizontal radius of two obstacles. The cyan circles are training queries while the black circle is one testing query. KR shows a very bad performance (see the blue bold dotted line). Our method produces suitable trajectories.

References

[1] S. Calinon, F. Guenter, and A. Billard, “On learning, representing and generalizing a task in a humanoid robot,” *IEEE Transactions on Systems, Man and Cybernetics, Part B. Special issue on robot learning by observation, demonstration and imitation*, vol. 37, no. 2, pp. 286–298, 2007.

[2] 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.

[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. [Online]. Available: http://dx.doi.org/10.1162/NECO_a_00393

[4] T. Matsubara, S.-H. Hyon, and J. Morimoto, “Learning parametric dynamic movement primitives from multiple demonstrations,” *Neural Netw.*, vol. 24, no. 5, pp. 493–500, Jun.

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

[6] J. Peters, S. Vijayakumar, and S. Schaal, “Reinforcement learning for humanoid robotics,” in *IEEE-RAS International Conference on Humanoid Robots (Humanoids2003)*, Karlsruhe, Germany, Sept.29-30, 2003.

[7] M. Toussaint and S. Vijayakumar, “Learning discontinuities with products-of-sigmoids for switching between local models,” in *Proceedings of the 22Nd International Conference on Machine Learning*, ser. ICML ’05. ACM, 2005, pp. 904–911.

[8] C. G. Atkeson, A. W. Moore, and S. Schaal, “Locally weighted learning,” *Artif. Intell. Rev.*, vol. 11, no. 1-5, pp. 11–73, Feb. 1997.

[9] H. Mueller, “Weighted local regression and kernel methods for nonparametric curve fitting,” *Journal of the American Statistical Association*, vol. 82, no. 397, pp. 231–238, 1987.

[10] T. Hastie and C. Loader, “Local regression: Automatic kernel carpentry,” *Statist. Sci.*, vol. 8, no. 2, pp. 120–129, 1993.

[11] E. A. Theodorou, J. Buchli, and S. Schaal, “Learning policy improvements with path integrals,” in *International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, 2010.

[12] T. Asfour, N. Vahrenkamp, D. Schiebener, M. Do, M. Przybylski, K. Welke, J. Schill, and R. Dillmann, “Armar-iii: Advances in humanoid grasping and manipulation,” *Journal of the Robotics Society of Japan*, vol. 31, no. 4, pp. 341–346, 2013.

[13] N. Vahrenkamp, M. Kröhnert, S. Ulbrich, T. Asfour, G. Metta, R. Dillmann, and G. Sandini, “Simox: A robotics toolbox for simulation, motion and grasp planning,” in *International Conference on Intelligent Autonomous Systems (IAS)*, 2012, pp. 585–594.

[14] N. Vahrenkamp, M. Wächter, M. Kröhnert, K. Welke, and T. Asfour, “The robot software framework armarx,” *Information Technology*, vol. 57, no. 2, pp. 99–111, 2015.