Reactive, Task-specific Object Manipulation by Metric Reinforcement Learning

Simon Hangl*, Emre Ugur*,
Sandor Szedmak* and Justus Piater*
Intelligent and Interactive Systems
University of Innsbruck
Email: *first.last@uibk.ac.at

Ales Ude†
Department of Automatics,
Biocybernetics and Robotics, Jozef Stefan Institute
Ljubljana
Email: †first.last@uibk.ac.at

***Abstract*—In the context of manipulation of dynamical systems, it is not trivial to design controllers that can cope with unpredictable changes in the system being manipulated. For example, in a pouring task, the target cup might start moving or the agent may decide to change the amount of the liquid during action execution. In order to cope with these situations, the robot should smoothly (and timely) change its execution policy based on the requirements of the new situation. In this paper, we propose a robust method that allows the robot to smoothly and successfully react to such changes. The robot first learns a set of execution trajectories that can solve a number of tasks in different situations. When encountered with a novel situation, the robot smoothly adapts its trajectory to a new one that is generated by weighted linear combination of the previously learned trajectories, where the weights are computed using a metric that depends on the task. This task-dependent metric is automatically learned in the state space of the robot, rather than the motor control space, and further optimized using using reinforcement learning (RL) framework. We discuss that our system can learn and model various manipulation tasks such as pouring or reaching; and can successfully react to a wide range of perturbations introduced during task executions. We evaluated our method against ground truth with a synthetic trajectory dataset, and verified it in grasping and pouring tasks with a real robot.**

I. Introduction

One open problem in robotics is to design controllers that are robust to the unexpected changes in the environment during complex object manipulation tasks. Even humans struggle to cope with this kind of unexpected changes; e.g. a tennis ball can change its predicted trajectory due to external forces such as wind while the tennis player already started striking the ball, forcing the player to adapt the strategy. There are several issues that have to be solved for handling a wide range of external disturbances while ensuring successful task achievement. First of all, the robot should quickly *adapt* to the new situation. For this, we propose to change the execution policy by switching to a new trajectory that is valid for the new observed situation. This raises the problem of how to formulate a control policy such that the method has the ability to *switch* between trajectories for different situations on the fly in reaction to extraneous changes in the system state. This has to be done in a smooth and robust way such that the robot does not undergo high velocities or forces which may cause danger to humans as well as to the robot itself. Another key aspect is the ability to *generalize* to novel situations, as it is not possible to provide training data for all situations that may occur. The policy should be able to generate trajectories which can be used to solve a problem in an unseen situation (e.g. different initial states).

978-1-4673-7509-2/15/$31.00 2015 IEEE

The contribution of this paper is a control policy model that meets these requirements (smooth adaptation and generalization) while still preserving conceptual simplicity. We first learn a database of execution trajectories that solves the task in a number of situations, i.e. system states. When encountered with an unseen situation, the model estimates a new trajectory by computing a weighted linear combination of these trajectories, where the weights depend on the task. Another major contribution is the automatic learning of these weights which makes the method applicable to several different manipulation tasks. The method is able to reuse arbitrary open loop controllers such as splines or Dynamic Movement Primitives (DMPs) [13], or even closed loop controllers. This makes it easy to reuse existing methods (and their advantages) with our new approach. The weighting of the individual database trajectories (and therefore their importance for the current situation) is measured using a metric on the environment state space. This utilizes the assumption that trajectories with similar initial conditions (environment states) have similar shape as well. Under this assumption, a metric measuring the similarity of the initial system state can be used to generalize to unseen situations. It cannot be assumed that a single metric can be used for any kind of manipulation problem. Therefore, to make the model applicable to a wide range of manipulation problems, we propose metric learning to determine an initial guess of the appropriate metric for a specific task. The automatic (unsupervised) learning of the task specific metric is one of the major contributions of this paper. Furthermore, we show how reinforcement learning (RL) can further improve the performance of the proposed metric learning algorithm. We argue that our system fits well with the popular control policy representations such as DMPs, and RL methods such as PI^2 [15], PoWER [6] and Policy Gradient Descent [12], which have shown good performance in robot applications. Therefore, our method has the advantage that it can easily be used with the current systems in robot labs without significant re-implementation effort. Additionally, our method uses a low number of open parameters that have to be learned, by assuming an implicit low dimensionality of the manipulation task (low-dimensional environment space).

In section VI we show that this indeed is the case for rather complex problems (e.g. pouring corn/water in a moving cup).

II. Related Work

A lot of research on motor learning in recent years focused on ways to adapt motor skills to external perceptual information, which is often related to the goal of an action. Perceptual queries and (local) regression methods are used to generate an appropriate dynamic motor primitive (DMP) from a database of example movements by learning a mapping from the environment state to the DMP parameters, which is executed afterwards (2-step process). [17], [3] use Gaussian Process Regression, while [2] uses a mix of ISOMAP and Support Vector Machines. A further extension of these methodis provided by [14], where they merged the 2-step process to a single step by making the DMP function approximator directly dependent on the environment state. An approach which is similar to the method proposed in this paper has been done by [8], where lower-level policies are used for given contexts, while a higher level policy generalizes in the context space. While DMPs can be used to encode a single motor primitive, in [5] Gaussian mixture models are used to encode multiple demonstrations within a single, stable dynamic system. Similarly, the method described in [9] encoded multiple demonstrations by shaping a parametric attractor landscape in a set of differential equations. A non-linear optimization problem had to be solved for this purpose.

Motor primitives can be generalized to new situations also by RL [7]. In contrast to these works, our research focuses not only on generalization to unseen situation, but considers reactive trajectory adaption explicitly as well. There is work on control policies that are able to react on unseen situation to some extent, such as DMPs [13]. However, these control policies can only react in a very limited way by stretching/compressing the trajectory or by changing the attractor of the system. On the other hand, the method described in this work is able to handle a wider range of reactive behaviours while still preserving advantages of control policies such as DMPs as it reuses existing models in a simple way.

The most similar method compared to the one proposed in this paper was developed by [10], where they use a weighted combination of database trajectories as well for the specific task of table tennis. This weighting is based on the current environment state. However, our approach is more general as they explicitly implement this weighting for the purpose of table tennis. The major question for an approach with weighted combination is how to determine the weights. In [10], they provided these weights by supervised learning, which makes it impossible to use it for other manipulation tasks. In our approach, one of the major contributions is automatic learning of a metric and therefore the weights for the trajectories, while still preserving the general powerfulness of a weighted combination approach.

III. Control Policy Model

The underlying idea of our method is to generate a database of primitive trajectories solving the same task in different environment states. The trajectories in the database can be raw measured data or trajectories encoded by more complex base control policies such as DMPs. This database is indexed by the environment state described by the *environment state* $\mathbf{q}_i$. This environment state is task dependent and should be selected such that it captures all the information that is required to perform the task. In a grasping domain this would be a library of grasping trajectories of the same object at different locations (environment states) $\mathbf{q} = (x_i, y_i)$.

Assuming that a slight change in the environment state also results in a slight change of the corresponding trajectory, one can *generalize to an unseen situation* (environment state) if the current environment state is within the space spanned by the samples in the database. However, depending on the task, it is not obvious how important each database trajectory is for the current situation (and therefore how strongly it should be weighted). Suppose that in the grasping domain the success of the grasp not only depends on the object location, but also on the object weight. Then the environment state would be given by (x_i, y_i, m_i), where m_i is the mass of the object. Not every component of this vector is of the same importance for the task (and some components may be correlated, e.g. the reaching position might be more important than the mass). We show that it is possible to determine the *relative importance* of each single database trajectory for the current situation. In our approach, we do this by learning a metric $m(\mathbf{q}_k, \mathbf{q}_l)$ which measures the distance between environment states. If the distance is high/low, the importance for the current situation is low/high. Generalization is done by interpolation where more important database samples are weighted more strongly.

The advantage of this approach is that in case the environment changes, it is feasible to re-weight the sample trajectories on the fly to achieve *adaptive behaviour*. This can be done by measuring the new environment state B and computing the new weight of each sample trajectory for the current situation. This information can be used to adapt to an unseen situation even if the current task is already under execution. In the grasping example, this would for example be a change of location. Assume that the initial object position is at $\mathbf{q}_A = (5,5)$. While executing the grasping motion, the object is moved to position $\mathbf{q}_B = (7,8)$. In our method, this only requires the re-estimation of the weights and a smooth transition $\mathbf{q}_A \rightarrow \mathbf{q}_B$. Note that in this specific application other control policies such as dynamic movement primitives can produce similar results, as this task only requires adapting the attractor point. However, in case generalization is required at a different point in time (e.g. in the middle of the execution), generalization and adaptive manipulation with DMPs is not straightforward. One example might be an obstacle avoidance task (see evaluation in section VI), where in the case of DMPs generalization to obstacle heights requires more complex algorithms such as a potential field approach [11]. Our method provides this functionality out of the box. On the other hand our approach is completely data-driven and requires more data to achieve the same task.

Our method strongly relies on solid importance estimation of the database trajectories, i.e. the metric which measures this importance. The first important observation is that the metric is task specific (the metrics for a reaching task and a pouring task will be different as the environment state spaces differ as well) which requires a metric learning algorithm. The biggest problem is the lack of ground truth distances between the environment states (which makes it impossible to use simple

regression). Therefore we propose to learn an initial guess of the metric by comparing the database trajectories by themselves instead of the environment states, assuming distances between trajectories directly map to distances between their corresponding environment states. Note that this assumption *does not hold in the other direction* in general, which is why this approach is only valid for determining an initial guess for the metric.

One further assumption is that a single metric is valid to measure the distance of vectors in the whole environment state space (see Table II for an evaluation of this assumption). This means that if it is possible to learn a valid metric for a single point within the environment state space, this metric is (approximately) valid for the whole space. Therefore, the initial guess can be learned by performing RL for a single environment state (this state should not be part of the database yet) to obtain a good metric estimate according to some task-specific reward function.

A. Trajectory Blending

We assume a database set Z of M sample trajectories associated with the environment state $\mathbf{q}_k$ (environment state in which the trajectory solves the task). There are no constraints on the vector $\mathbf{q}_k$, it is assumed that it intrinsically captures all required information about the environment to achieve the task (e.g. if the problem is a reaching task, the environment state would be a Cartesian goal coordinate vector). Hence, we define the trajectory database

$$Z = \big\{\mathbf{y}_{\mathrm{d}}^k(t_{k,j}), \dot{\mathbf{y}}_{\mathrm{d}}^k(t_{k,j}), \ddot{\mathbf{y}}_{\mathrm{d}}^k(t_{k,j}); \mathbf{q}_k \mid k = 1,\ldots,M;\ j = 1,\ldots,T_k\big\} \tag{1}$$

where $\mathbf{y}_{\mathrm{d}}^k(t_{k,j})$, $\dot{\mathbf{y}}_{\mathrm{d}}^k(t_{k,j})$ and $\ddot{\mathbf{y}}_{\mathrm{d}}^k(t_{k,j})$ are positions, velocities and accelerations sampled from measured trajectories. T_k is the number of points along trajectory k and $\mathbf{q}_k \in \mathbb{R}^N$ is the corresponding environment state. The choice of the environment state is essential for solving a specific task generalisation and the exact definition is up to the engineer. For each sample trajectory $\{\mathbf{y}_d^k(t_{k,j}),\ \dot{\mathbf{y}}_{,d}^k(t_{k,j}), \ddot{\mathbf{y}}_d^k(t_{k,j}) \mid j = 1,\ldots,T_k\}$, an arbitrary parametrized control policy $\pi_k(\mathbf{q}, t, \theta(t))$ can be learned by appropriate machine learning methods (e.g. regression). Here, $\theta(t)$ denotes the vector of control policy parameters (the learning method is policy specific). In our experiments, we used DMPs as base control policy model.

We propose a novel control policy model given by a linear combination of all trajectories in the database. Assuming Z can approximate the manifold of all solutions for a given task, an arbitrary task-specific trajectory $\mathbf{y}_{\mathrm{new}}$ solving the task for an unseen situation $\mathbf{q}_{\mathrm{new}}$ can be approximated by

$$\mathbf{y}_{\mathrm{new}}(\mathbf{q}_{\mathrm{new}}, t, \theta) = \frac{1}{s_{\mathrm{reg}}} \sum_{i=1}^{M} s_i \mathbf{y}_i(\mathbf{q}_i, t, \theta_i) \tag{2}$$

where s_{reg} is a normalization term given by $s_{\mathrm{reg}} = \sum_{i=1}^{M} s_i$ and $\mathbf{y}_i$ is the result of the action $\mathbf{u}_i(t) \sim \pi_i(\mathbf{q}_i, t, \theta_i(t))$. The velocity $\dot{\mathbf{y}}_{\mathrm{new}}$ and acceleration $\ddot{\mathbf{y}}_{\mathrm{new}}$ can be derived from the positions, as the time steps t are given (e.g. using a KUKA LWR only the positions at given time steps are submitted to the robot). The computation of the coefficients s_i is non-trivial in general, as it highly depends on the task. Given the assumption that similar situations can be solved by similar trajectories, it is reasonable compute the coefficients s_i with some similarity measure on the environment state space. The coefficients should have high values whenever a trajectory is more important than others for solving the current task. As it is assumed that the situation is fully described by the environment state vector, the coefficients can be defined by any strictly (with distance) decreasing function like

$$s_i(\mathbf{q}_{\mathrm{new}}, \mathbf{q}_i) = \exp\left(-\alpha_{\mathrm{m}} m(\mathbf{q}_{\mathrm{new}}, \mathbf{q}_i)^2\right) \tag{3}$$

where $m : Q \times Q \to \mathbb{R}^+$ denotes an arbitrary metric and α_{m} is a free parameter measuring how strongly distant trajectories should influence the result. This parameter can be either hand tuned or learned in the RL step (section IV-C).

As the environment states $\mathbf{q}$ can have arbitrary dimensions and units, the metric has to be learned automatically for each task. One subclass of metrics is given by a metric parametrized like the *Mahalanobis distance*, defined by

$$m(\mathbf{q}_1, \mathbf{q}_2) = \sqrt{(\mathbf{q}_1 - \mathbf{q}_2)^{\mathrm{T}} \mathbf{M} (\mathbf{q}_1 - \mathbf{q}_2)} \tag{4}$$

where $\mathbf{M}$ is positive semi-definite. It is important to note that with our proposed control policy formulation it is possible to *generalize* to unseen situations $\mathbf{q}_{\mathrm{new}}$ by computing the coefficients s_i (equation 3), given the learned metric (Section IV). A simplified version of the proposed control policy is summarized in Algorithm 1. If the environment changes during execution, the coefficients can be re-estimated and smoothly transitioned from old to new values (Section V).

Data: Control Policies π_k, Corresponding environment states $\mathbf{q}_k$, Metric $m(\mathbf{q}_i, \mathbf{q}_j)$, Current environment state $\mathbf{q}_c$
Result: Executed trajectory
Current time $t_c \leftarrow 0$;
initialize weights by $s_i \leftarrow \exp\left(-\alpha_{\mathrm{m}} m(\mathbf{q}_c, \mathbf{q}_i)^2\right)$;
while $t_c < t_{max}$ **do**
  $\mathbf{q}_c \leftarrow$ current environment state;
  if $\mathbf{q}_c$ *changed* **then**
    Recompute weights (smooth transition) by $s_i \leftarrow \exp\left(-\alpha_{\mathrm{m}} m(\mathbf{q}_c, \mathbf{q}_i)^2\right)$;
  end
  $\pi(t_c) \leftarrow \frac{1}{s_{\mathrm{reg}}} \sum_{i=1}^{M} s_i \pi_i(t_c)$;
  Execute $\pi(t_c)$;
  $t_c \leftarrow t_c + \Delta t$
end

Algorithm 1: Control Policy Execution

IV. Metric Learning

Learning Mahalanobis-style distances in high-dimensional environment spaces has been studied extensively in computer vision and recommender systems [1]. However, these approaches tend to require more samples than are available in the robotics domain. One major contribution of this paper is to provide an algorithm to learn a Mahalanobis-style metric automatically from the database Z. We seek to learn a metric for $m(\mathbf{q}_i, \mathbf{q}_j) = m_{ij}$, where $\mathbf{q}_i, \mathbf{q}_j$ are environment states and m_{ij} is the distance between these vectors. In order to do this in a supervised fashion, the values $\mathbf{q}_i$, $\mathbf{q}_j$, m_{ij} are required. A summary of the algorithm can be found in algorithm 2.

Data: Control Policies π_k, Corresponding environment states $\mathbf{q}_k$, Reward function $r()$
Result: Metric $m(\mathbf{q}_i, \mathbf{q}_j)$
Compute $m_{ij} \leftarrow d(\mathbf{q}_i, \mathbf{q}_j)$ for all (i, j) in database;
Compute initial guess for $\mathbf{M}_0$ by equations 8, 9;
Choose q_{rl} randomly in environment state space;
Apply PoWER: $\mathbf{M} \leftarrow \text{PoWER}_{\mathbf{M}}(\pi(\mathbf{q}_{rl}, \mathbf{M}_0), r)$;

Algorithm 2: Metric Learning

A. Metric Initialization

To approximate the values m_{ij}, we propose to *compare the trajectories themselves* instead of comparing the environment states. Therefore a similarity measure on trajectories needs to be defined (see [19] for a comprehensive review of trajectory comparison methods). We propose to use an adapted version of the trajectory similarity measure in [4] where the Euclidean distance between two F-dimensional trajectories $\mathbf{a}, \mathbf{b}$ is given by

$$d(\mathbf{a}, \mathbf{b}) = \frac{1}{F} \sum_{n=1}^{F} \frac{c_n}{T} \sum_{m=1}^{T} (a_m^n - b_m^n)^2. \tag{5}$$

F is the number of degrees of freedom and a_m^n and b_m^n are the joint positions of the n-th degree of freedom at time step m. The free parameters c_n denote the influence of the specific degree of freedom on the final distance. These coefficients highly depend on the robot and on the space in which the trajectory is defined (e.g. Cartesian space or Joint space) and on the manipulation task as well (not all degrees of freedom are of the same importance). If the importance is not known (as in our experiments), the weights can be defined as $c_n = 1$. As these values are only used for an initial guess of the metric, the method is not sensitive to the selection of these weights. Both trajectories are re-sampled such that samples at the same time step are compared.

B. Supervised Metric Learning

Supervised metric learning is the problem of deriving the coefficients of some metric model when a set of vector pairs and their distance is given. In case the environment state space is low-dimensional (which might be the case for many manipulation problems), we propose to learn a metric that is parametrized like the Mahalanobis distance (we will further refer as *Mahalanobis-style* (MHS) distance), by reusing ideas from *multidimensional scaling* (MDS) [18].

MDS is a method to reconstruct a matrix $\mathbf{Q}$ of coordinates if the matrix $\mathbf{D}$ of pairwise squared distances between these vectors is known. From the output of this procedure and by knowing the vectors $\mathbf{q}$ already, the MHS distance can be estimated by computing the Moore-Penrose Inverse, as we describe next. Let $\mathbf{D} \in \mathbb{R}^{K \times K}$ be the matrix containing the squared distances of the pairs $(\mathbf{q}_i, \mathbf{q}_j)$ for all $i, j \leq K$, where K is the number of environment states. The matrices $\mathbf{D}, \mathbf{Q}$ are defined by

$$\mathbf{Q} = \begin{pmatrix} \mathbf{q}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{q}_K^{\mathrm{T}} \end{pmatrix}, \quad \mathbf{D} = \begin{pmatrix} m_{11}^2 & \dots & m_{1K}^2 \\ \vdots & \ddots & \vdots \\ m_{K1}^2 & \dots & m_{KK}^2 \end{pmatrix}. \tag{6}$$

Multidimensional scaling can only be applied in case the matrix $\mathbf{M}$ of the MHS metric is an identity matrix, which is not the case in general. Therefore, we use the identity $m_{ij}^2 = m(\mathbf{q}_i, \mathbf{q}_j) = \mathbf{q}_i^T \mathbf{M} \mathbf{q}_j = \mathbf{q}_i^{\mathrm{T}} \mathbf{Z}^{\mathrm{T}} \mathbf{Z} \mathbf{q}_j = \mathbf{k}_i^{\mathrm{T}} \mathbf{k}_j$ to define the feature vectors $\mathbf{y}_i = \mathbf{Z}\mathbf{q}_i$, $\mathbf{k}_j = \mathbf{Z}\mathbf{q}_j$ (note that the feature vectors $\mathbf{k}$ are not the same as the trajectory vectors $\mathbf{k}$ in equation 1). This can be done since the matrix $\mathbf{M}$ is positive semi-definite and therefore can be decomposed as $\mathbf{M} = \mathbf{Z}^{\mathrm{T}}\mathbf{Z}$. With this feature vector, we can apply multidimensional scaling [16]. The cosine law is valid for vector spaces with standard inner product and can therefore be used with the feature mapping. Assuming a triangle with corners i, j, k, the angle θ_{jik} between edges (i,k) and (i,j) is given by

$$\cos\theta_{jik} = \frac{m_{ij}^2 + m_{ik}^2 - m_{jk}^2}{2 m_{ij} m_{ik}}. \tag{7}$$

Further, we define $b_{jik} := m_{ij} m_{ik} \cos\theta_{jik} = \mathbf{k}_i^T \mathbf{k}_j$, which is the scalar product of vectors $\mathbf{k}_i$ and $\mathbf{k}_j$ within the triangle (i, j, k). Therefore, the matrix B_i can be defined as

$$B_i = \mathbf{K_i}^{\mathrm{T}} \mathbf{K_i} \tag{8}$$

for some arbitrary i, where $\mathbf{K_i}$ is a matrix containing the vectors $\mathbf{y}_j$ for all j excluding the vector $\mathbf{k}_i$. This matrix can be computed by equation 7. As the known matrix $\mathbf{B}_i$ (derived from the sample distances in the previously-described way) is symmetric, it can be decomposed by singular value decomposition (SVD) as $\mathbf{B}_i = \mathbf{U}\mathbf{V}\mathbf{U}^{\mathrm{T}}$, which yields the solution $\mathbf{K_i} = \mathbf{V}^{1/2}\mathbf{U}$. Finally, by knowing the matrix $\mathbf{Q}$, we can derive $\mathbf{M} = \mathbf{Z}^{\mathrm{T}}\mathbf{Z}$ from

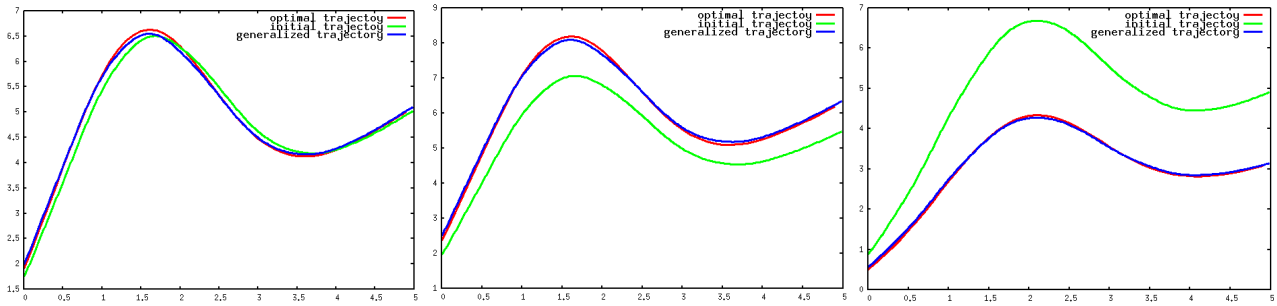
$$\mathbf{Z} = \mathbf{K_i}^{\mathrm{T}} \mathbf{Q_i} \left(\mathbf{Q_i}^{\mathrm{T}} \mathbf{Q_i} \right)^{-1}. \tag{9}$$

C. Metric Reinforcement Learning

The initial estimate computed in the previous section can be refined using policy reinforcement learning methods such as PI^2 [15], PoWER [6] or policy gradient descent methods [12]. However, these methods cannot be applied to refine the coefficients of the matrix $\mathbf{M}$ without any adaptation, as they do not preserve the metric properties. By using the property that the positive semi-definite matrix $\mathbf{M}$ can be rewritten as $\mathbf{M} = \mathbf{Z}^{\mathrm{T}}\mathbf{Z}$, these properties can be preserved, by performing the updates on $\mathbf{Z}$. The matrix $\mathbf{Z}$ can be computed by SVD of $\mathbf{M} = \mathbf{U}\mathbf{V}\mathbf{U}^{\mathrm{T}}$, where $\mathbf{U}$ is a square matrix and $\mathbf{V}$ is a diagonal matrix. Then $\mathbf{Z}$ is given by

$$\mathbf{Z} = \mathbf{U}^{\mathrm{T}} \mathbf{V}^{1/2}. \tag{10}$$

$\mathbf{Z}$ can be modified without sacrificing the positive semi-definiteness of $\mathbf{M}$. To improve the metric, a single random environment state $\mathbf{q}_{rl}$ from the environment state space has to be chosen. The only constraint is that it should be neither at the border of the environment state space nor already be stored in the database. If this is ensured, reinforcement learning can be performed in this state by optimizing the upper triangle coefficients of the symmetric matrix $\mathbf{Z}$ (free policy parameters) and copying these values to the lower triangle such that it is symmetric again. These coefficients are changed by RL and a new roll-out is then performed by Algorithm 1. Note that the cost function $r(\pi)$ has to be given by the user. In Section VI, PoWER is used (Algorithm 2).



(a) $\mathbf{q} = (1.4, 5.1)$ (b) $\mathbf{q} = (1.4, 6.3)$ (c) $\mathbf{q} = (1.9, 3.1)$

Fig. 1. Synthetic data experiments: Generalization performance in simulation for selected environment states. Red trajectories show the ground truth; Green trajectories show the initial guess by using trajectory comparison; Blue trajectories show the final proposed generalization after reinforcement learning

V. Smooth Trajectory Switching

With the trajectory blending in described in equation 2, the policy provides a mechanism for *adaptive manipulation* if the environment state changes during the execution. The agent can continuously update the current environment state $\mathbf{q}_{\text{next}}$ and re-estimate the coefficients s_i^{next}. Directly replacing the current coefficients by the new ones would yield dangerous accelerations. Therefore, the transition has to be done smoothly. To be more explicit, let

$$S_{\text{curr}} = S(\mathbf{q}_{\text{curr}}, t_0) = \left\{ s_i^{t_0}(\mathbf{q}_{\text{curr}}, \mathbf{q}_i, t_0) \mid i = 1, \ldots, M \right\} \quad (11)$$

be the set of coefficients for the current environment state $\mathbf{q}_{\text{curr}}$ at time t_0, and $\mathbf{q}_i$ the environment states of all trajectories in the database. A smooth transition between the current state and the new state is done by adapting the coefficients s_{i} used in equation (2) as

$$s_i^{t_0+\Delta t} = s_i^{\text{next}} \left(1 - \mathrm{e}^{-\alpha_s \Delta t}\right) + s_i^{t_0} \mathrm{e}^{-\alpha_s \Delta t} \quad (12)$$

with the free parameter α_s, which determines the speed of the adaptation. Typical values for α_s are between 0.5 and 1.5. It is straightforward to see that

$$\lim_{\Delta t \to 0} s_i^{t_0+\Delta t} = s_i^{t_0}, \quad \lim_{\Delta t \to \infty} s_i^{t_0+\Delta t} = s_i^{t_{\text{next}}}, \quad (13)$$

which means that the current weight $s_i^{t_0}$ vanishes, while $s_i^{t_{\text{next}}}$ becomes dominant for large Δt.

VI. Evaluation

In the first experiment, we analysed the generalization performance of the proposed method along with an evaluation of the adaptive mechanism using synthetic, 1-dimensional trajectories that model obstacle avoidance. The second experiment is a reaching task with a 7-DoF KUKA LWR robot (see Fig. 4), where the underlying optimal metric on the environment state space was already known. Finally, a more involved pouring task was realized, where a predefined amount of corn is poured from one cup into another while the target cup was moving.

A. Evaluation by Synthetic Data

The first task was a synthetic 1D obstacle avoidance task with a 2D environment space (obstacle of height h at location p). No real robot was used in this example; this is a toy example to evaluate the generalization performance in a task where ground truth (given by an analytical function) is known. Further we evaluated the assumption of a globally valid metric (valid for the whole environment state space) and the adaptive mechanism.

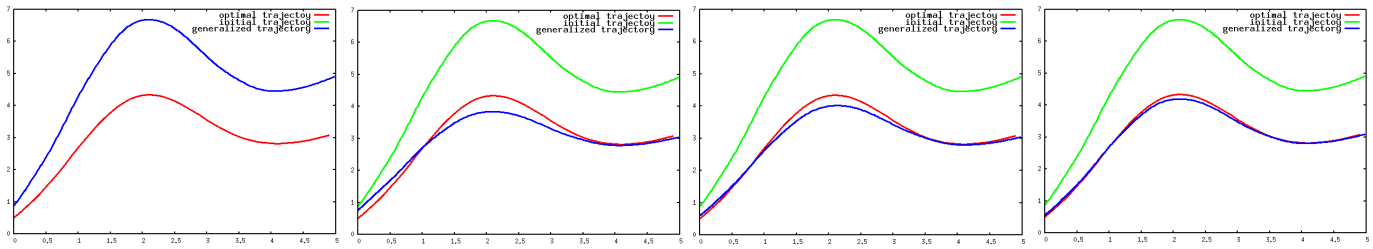
Experimental Setting: The obstacle avoidance trajectory is modelled by an adapted Gaussian function where the obstacles are located at the maximum (height given by amplitude h)

$$y(t) = h\mathrm{e}^{-(t-p)^2/2} + 0.2ht \quad (14)$$

where the environment state is given by $\mathbf{q} = (p, h)$. The function is chosen such that it avoids an obstacle of height h at location p. The advantage of this approach is straightforward database generation and the availability of ground truth. The database included 18 sample trajectories that were generated from the set $\mathbf{q}_{ph} = (p, h) \in \{0.5, 0.9, 1.3, 1.7, 2.1, 2.5\} \times \{3, 5, 7\}$. We used DMPs as control policy with 16 Gaussian basis functions. RL (PoWER) was performed with 8 roll-outs per update step. The reward function used for reinforcement learning was defined by $r(\pi) = 1/\exp(m(\mathbf{y}, \mathbf{y}_{\text{des}}))$ where $m(\mathbf{y}_i, \mathbf{y}_j)$ is the trajectory metric described in Section IV-A, and $\mathbf{y}$ is the trajectory generated by our proposed metric policy π. Therefore, the highest reward will be given if the method recreates the ground truth trajectory.

Results: Fig. 1 shows the generated trajectories for the simulated obstacle avoidance task, where each plot corresponds to a different unseen environment state, i.e. obstacle configuration in terms of its position and height. The green line visualizes the prediction of the trajectory by using the metric initialization algorithm in Section IV-A, while the red line shows the ground truth trajectory. It can be seen that the initialization does not provide any guarantees that the executed trajectory will be appropriate for the current situation (e.g. Fig. 1(c)). However, in many cases it provides a good starting point for the rest of the algorithm as can be seen in Figs. 1(a) and 1(b) (both close to optimum).

Table I provides details about the convergence behaviour of the reinforcement learning step described in Section IV-C. The first column lists the environment state on which the reinforcement learning was performed to learn the global metric. Please note that in a practical setting this learning only has to be done for one single environment state within the space to estimate the metric. Because of the intrinsic low dimensionality of the problem, the reinforcement learning



(a) Update step 1: Initial trajectory coincides with generalized trajectory (b) Update step 2 (c) Update step 4 (d) Update step 6

Fig. 2. Reinforcement Learning: Convergence behaviour for constant environment state $\mathbf{q} = (1.9, 3.1)$ through RL update steps; red (ground truth), green (generated trajectory *before* RL), blue (generated trajectory at time step i)

TABLE I. SYNTHETIC DATA EXPERIMENTS: REINFORCEMENT LEARNING PERFORMANCE FOR SELECTED ENVIRONMENT STATES

environment state	Initial Reward	Final Reward	Required Updates
(0.6, 6.3)	0.42	0.82	5
(1.4, 5.1)	0.84	0.94	7
(1.4, 6.3)	0.84	0.91	9
(1.5, 4.5)	0.64	0.95	9
(1.9, 3.1)	0.16	0.96	6
(1.9, 6.3)	0.28	0.92	9

converged within less than 10 update steps. An example of the convergence behaviour is shown in Fig. 2. It can be seen that the method strongly improves already after the first parameter update in Fig. 2(b) and converges within 4 more update steps to a local maximum (see Fig. 2(d)).

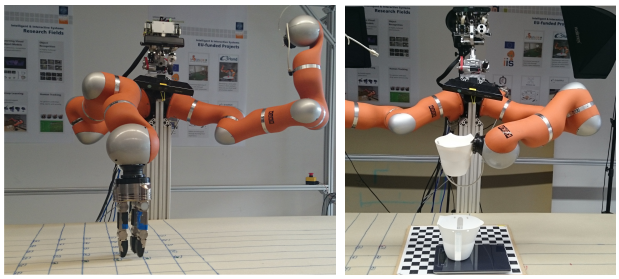
Additionally, the good generalization results should be emphasized, as all trajectories are executed with a reward of more than 0.82 (where the maximum reward is 1.0). Note that as this experiment is done with synthetic data, there is no meaningful way to measure the error; we selected the reward to measure if the ground truth trajectory was reproduced.

We next evaluated how adaptive our system is in the face of external perturbations by changing the position and height of the obstacle on the fly during trajectory execution where both the initial and changed configurations were previously unseen. The adaptive manipulation mechanism is illustrated in Fig. 3, which shows a switch from ground truth trajectories 1 (green, h = 6.0) to 2 (blue, $h = 7.0$) at 1.5 seconds. Both trajectories were not in the database before. Note that the task-specific trajectory shape (red) is still preserved while switching from green to blue. Further, it is possible to set the transition speed - Fig. 3(a) shows a very smooth transition ($\alpha_s = 0.8$). However, if the environment changes are fast, one might need to react faster as well (Fig. 3(b), $\alpha_s = 1.3$). In Fig. 3(c) the smooth transitioning is disabled. This results in very abrupt jumps which is not a desirable behaviour (dangerous movement for humans and the robot). Finally, it was evaluated whether the learned metric is

TABLE II. SIMULATED DATA EXPERIMENTS: PERFORMANCE FOR UNSEEN ENVIRONMENT STATES

environment state	Reward
(1.4, 5.1)	0.92
(0.6, 6.3)	0.93
(0.5, 3.0)	0.83
(2.5, 7.0)	0.77
(1.5, 5.5)	0.93

only valid for a limited region around the environment state the reinforcement learning was performed on, or if it can be used



(a) Reaching Experiment (b) Pouring Experiment

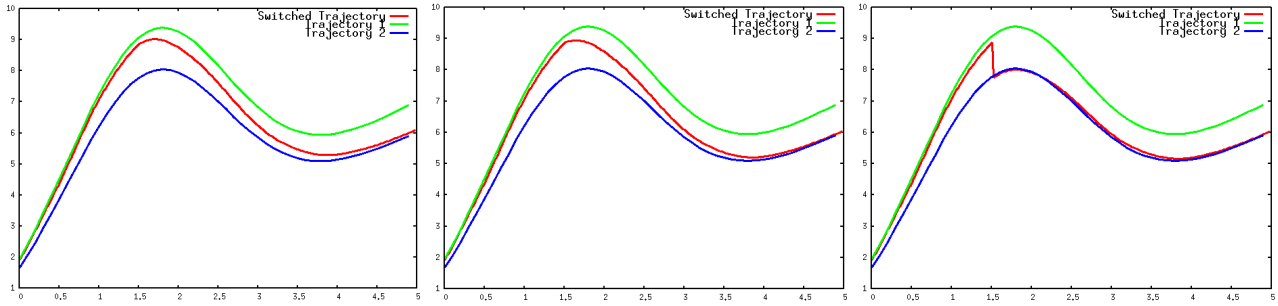
Fig. 4. Experiment setting of real-robot experiments

globally (Table II). The metric is learned by the reinforcement learning approach for the randomly selected environment state $\mathbf{q}_{learn} = (1.4, 5.1)$ and the performance of this metric with other environment states was tested Three out of five tests had a reward $r \geq 0.92$, while reduced performance ($r \approx 0.77$) was only observed for environment states that were located at the database border, e.g. $\mathbf{q} = (0.5, 3.0)$. As the method needs data around the requested environment state to be able to do a proper interpolation, reduced performance at the borders is not a problem.

B. Reaching task with the real robot

In this experiment, the environment state space as well as the cost function are straightforward to define. Therefore, the metric has an intuitive form and the result for the learned metric can be checked for plausibility of the results. Further, it is possible to perform evaluation in simulation as well as with a real-world robot in an automated way, as the performance is only given by the final reaching position, which made it feasible to sample the environment state space densely and to measure the performance with high precision.

Experimental Setting: The experiments were performed with a KUKA LWR 4+ (see Fig. 4(a)). The environment state space was the 2D reaching position on the table surface. Sample trajectories were provided by kinesthetic guiding for the environment states $\mathbf{q}_{xy} = (x, y) \in \{0.6, 0.7, 0.8, 0.9, 1.0\} \times \{0.5, 0.6, 0.7\}$ m. A DMP was learned for each sample. Note that these were learned in joint space; otherwise the reaching movement can simply be done by a simple point-to-point movement in Cartesian space, whereas in joint space the goal state and trajectory would require forward kinematics which was not used in the experiment. The trajectories had a length



(a) $\alpha_s = 0.8$ (b) $\alpha_s = 1.3$ (c) $\alpha_s \to \infty$ - Smooth switching disabled

Fig. 3. Switching from green to blue trajectory for synthetic obstacle avoidance task

of up to 14 seconds, which required 30 basis functions. RL in simulation was sufficient as the cost function only consists of the final Cartesian position. The cost function used for the trajectory generated by the metric policy π with Cartesian goal position $\mathbf{g}$ and desired goal position $\mathbf{g}_{\text{des}}$ was defined as $r(\pi) = 1/\exp\|\mathbf{g}_\pi - \mathbf{g}_{\text{des}}\|$. Per update, 5 roll-outs were executed in a simulator.[1] While the learning was done in the simulator, the performance was measured with the robot.

Results: In total 83 environment states were executed uniformly over the whole environment state space. The resulting average generalization error was 0.013 m $\pm$ 0.009 m. As baseline method we selected a nearest neighbour approach, where the average error was 0.036 m $\pm$ 0.017 m. It should be noted that without knowing the metric, it would not be possible to use the nearest-neighbour approach for an arbitrary problem (e.g. screwing a nut). However, because the trajectory database was indexed by the 2D reaching position, for this specific problem it was possible to select the nearest neighbour automatically. Additionally, the resulting metric was very intuitive, as the environment state space was the Cartesian space, where $\mathbf{M}$ should be the identity matrix. Indeed, the learned metric was

$$\mathbf{M} = \begin{pmatrix} 1 & -0.1698 \\ -0.1698 & 1.1672 \end{pmatrix}. \quad (15)$$

C. Pouring task with the real robot

A pouring task was performed, where m kg of corn had to be poured from cup A to cup B at position x on the table. As the environment state has components with differing units (the amount to pour is measured in kg and the location of cup B in meters) the metric cannot be intuitively guessed. Further, in this case the learning had to be done with the real robot as well. We also used our approach to show the power of our reactive control policy by updating the target cup's position online during trajectory execution.

Experimental Setting: Cup A was mounted on the end-effector of the KUKA LWR 4+, while cup B was placed on a line on the table surface (Fig. 4(b)). Cup A was filled with a fixed amount of corn. The environment state was chosen as $\mathbf{q} = (m, x)$. Sample trajectories were provided by kinesthetic guiding for 12 environment states distributed non-uniformly over the space with $0.044 \leq m \leq 0.38$ kg and $0.9 \leq x \leq 1.3$ m. The pouring success was measured with a balance located below cup B. The sample trajectories had a length of at most 35 seconds, which required 74 DMP basis functions. The reward function for reinforcement learning was selected as $r(\pi) = 1/\|m_{\text{des}} - m_{\text{mes}}\|$, where m_{des} is the desired poured amount and m_{mes} is the measured amount of corn respectively. 5 samples per update step were rolled out. For the adaptive manipulation, the left arm in figures 5 was moved during execution. The position data of the left arm was fed to the system directly from the robot control software with an update rate of 10 Hz to update the environment state.

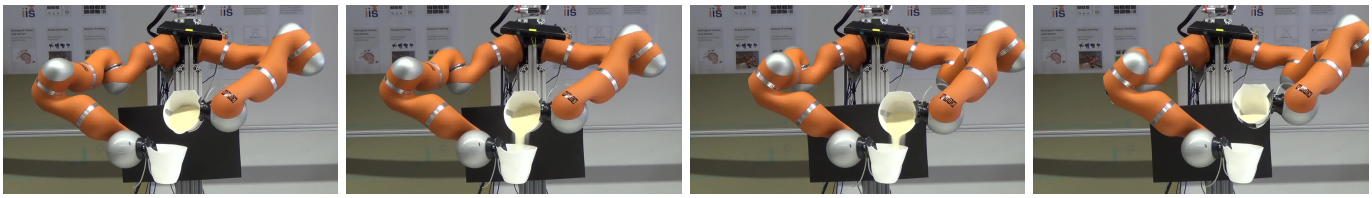
Results: In total, 17 samples were performed (randomly distributed over the environment state space), with a mean error of 0.0216 kg $\pm$ 0.0171 kg. Again, we compared our method to the nearest neighbour as baseline method, where the nearest neighbour was selected such that the poured amount of corn was optimal. This required execution of several possible sample trajectories and selecting the best (average error of 0.0425 kg $\pm$ 0.401 kg). This high standard deviation reflects the observation that nearest neighbour worked very well around sample points, while it failed in between (e.g. all or a significant amount of corn was spilled on the table), which is a highly undesirable behaviour. Contrary to this, our proposed method spilled at most 0.003 kg in 2 cases. We also evaluated the reactive behaviour of our approach by varying the x component of the environment state within the range of 0.9 to 1.3 m (moving cup B) while pouring 0.3 kg of corn. This experiment was repeated 3 times, where each trial succeeded with a poured amount of corn between 0.291, 0.295 and 0.3 kg respectively[2].

VII. Conclusions

We proposed a novel data-driven control policy method, which *generalizes* to unseen situations and *adapts* to environment changes. This is done by maintaining a task-specific trajectory database indexed by (low-dimensional) environment states. When a new situation is observed, these trajectories are weighted according to their importance, which is done by using a learned metric. It measures the distance (importance) of the environment states in the database. This metric is learned by using an initial guess and optimizing it by reinforcement learning for a *single* environment state.

[1] For the reinforcement-learning roll-outs we used the V-Rep robot simulation platform with the Bullet physics engine. As the cost function only consists of the final Cartesian position, the simulated kinematics of the robot was sufficient. Note that this only has been done because here it was not required to run the experiment in the real world. In general, 2D query space problems can be learned in real robot experiments as well (see pouring).

[2] https://iis.uibk.ac.at/public/shangl/pouring.mp4



(a) Pouring starts (b) Left arm moving right (c) Left arm moving right (d) End of pouring

Fig. 5. Adaptive Manipulation in a Pouring Task

We evaluated our method in three different experiments: synthetic obstacle avoidance, real-robot reaching and real-robot pouring. The plausibility of the method's base assumptions was tested. The most restrictive assumption is that small changes of the environment state result in small changes of the trajectory, which holds for many manipulation problems. However, the reader should be aware of the fact that this does not hold in general. Further, the metric is assumed to be valid for the whole environment state space, and the trajectories within the database are assumed to span the subspace of trajectories required for solving the task. This problem can be solved to some extent by using base control policies that are able to add a certain amount of generalization (e.g. DMPs as base policies can be used to generalize in task where attractor point generalization is sufficient). The third basic assumption is that the problem's intrinsic environment state is low-dimensional. This assumption holds in many real-world manipulation problems, as a low number of important properties can be easily identified by humans in many cases. However, in the current version of the approach the state vector has to be selected by the user. Future work will investigate automatic selection of the required data.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Communitys Seventh Framework Programmes FP7/20072013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreement no. 610532, Squirrel and FP7/2007-2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreement no. 270273, Xperience.

REFERENCES

[1] A. Bellet, A. Habrard, and M. Sebban. A survey on metric learning for feature vectors and structured data. *CoRR*, abs/1306.6709, 2013.

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

[3] D. Forte, A. Gams, J. Morimoto, and A. Ude. On-line motion synthesis and adaptation using a trajectory database. *Robotics and Autonomous Systems*, 60(10):1327–1339, 2012.

[4] Z. Fu, W. Hu, and T. Tan. Similarity based vehicle trajectory clustering and anomaly detection. In *ICIP (2)*, pages 602–605. IEEE, 2005.

[5] S. M. Khansari-Zadeh and A. Billard. Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957, 2011.

[6] J. Kober and J. Peters. Policy search for motor primitives in robotics. *Mach. Learn.*, 84(1-2):171–203, July 2011.

[7] J. Kober, A. Wilhelm, E. Oztop, and J. Peters. Reinforcement learning to adjust parametrized motor primitives to new situations. *Auton. Robots*, 33(4):361–379, 2012.

[8] A. Kupcsik, M. Deisenroth, J. Peters, and G. Neumann. Data-efficient generalization of robot skills with contextual policy search. In *Proceedings of the 27th National Conference on Artificial Intelligence (AAAI 2013)*, 2013.

[9] T. Matsubara, S.-H. Hyon, and J. Morimoto. Learning parametric dynamic movement primitives from multiple demonstrations. In *Proceedings of the 17th International Conference on Neural Information Processing: Theory and Algorithms - Volume Part I*, ICONIP'10, pages 347–354, Berlin, Heidelberg, 2010. Springer-Verlag.

[10] K. Mülling, J. Kober, O. Kroemer, and J. Peters. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279, 2013.

[11] D.-H. Park, H. Hoffmann, and S. Schaal. Combining dynamic movement primitives and potential fields for online obstacle avoidance. In *Adaptive Motion of Animals and Machines (AMAM)*, 2008.

[12] J. Peters and S. Schaal. Policy gradient methods for robotics. In *Proceedings of the IEEE International Conference on Intelligent Robotics Systems (IROS 2006)*, 2006.

[13] S. Schaal. dynamic movement primitives - a framework for motor control in humans and humanoid robots. In *the international symposium on adaptive motion of animals and machines*, 2003.

[14] F. Stulp, G. Raiola, A. Hoarau, S. Ivaldi, and O. Sigaud. Learning compact parameterized skills with a single regression. In *IEEE-RAS International Conference on Humanoid Robots*, 2013.

[15] E. Theodorou, J. Buchli, and S. Schaal. A generalized path integral control approach to reinforcement learning. *J. Mach. Learn. Res.*, 11:3137–3181, Dec. 2010.

[16] W. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17(4):401–419, 1952.

[17] A. Ude, A. Gams, T. Asfour, and J. Morimoto. Task-specific generalization of discrete and periodic dynamic movement primitives. *Robotics, IEEE Transactions on*, 26(5):800 –815, oct. 2010.

[18] F. W. Young. *Multidimensional Scaling: History, Theory, and Applications*. Lawrence, Erlbaum Associates, Publishers (Hillsdale, New Jersey; London), 1987.

[19] Z. Zhang, K. Huang, and T. Tan. Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes. In *Proceedings of the 18th International Conference on Pattern Recognition - Volume 03*, ICPR '06, pages 1135–1138, Washington, DC, USA, 2006. IEEE Computer Society.