Generative adversarial training of product of policies for robust and adaptive movement primitives

Emmanuel Pignat
Idiap Research Institute and EPFL
Switzerland
emmanuel.pignat@gmail.com

Hakan Girgin
Idiap Research Institute and EPFL
Switzerland
hakan.girgin@idiap.ch

Sylvain Calinon
Idiap Research Institute and EPFL
Switzerland
sylvain.calinont@idiap.ch

Abstract: In learning from demonstrations, many generative models of trajectories make simplifying assumptions of independence. Correctness is sacrificed in the name of tractability and speed of the learning phase. The ignored dependencies, which often are the kinematic and dynamic constraints of the system, are then only restored when synthesizing the motion, which introduces possibly heavy distortions. In this work, we propose to use those approximate trajectory distributions as close-to-optimal discriminators in the popular generative adversarial framework to stabilize and accelerate the learning procedure. The two problems of adaptability and robustness are addressed with our method. In order to adapt the motions to varying contexts, we propose to use a product of Gaussian policies defined in several parametrized task spaces. Robustness to perturbations and varying dynamics is ensured with the use of stochastic gradient descent and ensemble methods to learn the stochastic dynamics. Two experiments are performed on a 7-DoF manipulator to validate the approach.

Keywords: learning from demonstration, generative adversarial models, movement primitives, product of experts

1 Introduction

Adaptability and ease of programming are key features necessary for a wider spread of robotics in factories and everyday assistance. Learning from demonstrations (LfD) is an approach to address this problem. It aims to develop algorithms and interfaces such that a non-expert user can teach the robot new tasks by showing examples. In LfD, movements are commonly represented using movement primitives (MPs). They are used as building blocks of more complete skills in which they can be combined sequentially or simultaneously.

In LfD, the parameters of MPs are learned from a set of demonstrations. Ideally, motions synthesized from the MPs should match the distribution of demonstrations with the same variability [1, 2]. It can be later exploited for multiple usages, such as including additional constraints or objectives. Furthermore, keeping the variability is primordial when the demonstrations are used to initialize policy search [3]. Another desired feature of MPs is their adaptation to new situations or targets, such as moving objects. To that end, a common approach is to learn MPs in multiple parametric task spaces[1] [4, 5, 6]. For example, the task spaces can be attached to objects of interest [7, 8, 4] such that the movements are analyzed under several coordinate systems. However, for computational reasons, many of these approaches make an assumption of independence between the MPs in the different

[1]In this work, task spaces are not limited to the position and orientation of the end-effector but are general transformations of the configuration space. The configuration space itself will be considered as a task space with an identity transformation.

spaces; those are learned independently and only combined at the controller level, which results in distortions in the synthesis.

A consistent framework for learning multiple models jointly is the product of experts (PoE) [9, 10]. Models with unnormalized likelihood like PoEs are used in robotics for inverse optimal control (IOC) [11, 12, 13]. However, they either rely on expensive approximations of the normalizing constant [12] or learn only weights of predefined features [13]. Generative adversarial modelling [14] has been proposed as a more efficient approach with improved training stability [15, 16, 17].

In this work, we propose to train MPs within the generative adversarial framework which we call generative adversarial movement primitives (GAMP). We propose several adaptations of the discriminator and a particular parametrization of the policy to meet the requirements of LfD; as performing demonstrations on the physical systems is costly, we typically only have a few trajectories (from 5 to 20 depending on the complexity). Also, the training process should be interactive and thus relatively fast (from a few seconds to a few minutes). The proposed approach can be classified as model-based imitation learning [2]. Given a prior knowledge about the dynamics of the system, it uses model-based policy search to minimize an imitation cost. As we will show through real robot experiments, rough dynamic models make the process sample-efficient. We also propose a variant of the method to refine these models through executions on the real system. Our approach treats both epistemic uncertainty (coming from partial knowledge of the system) and aleatoric uncertainty (coming from stochasticity of the system), resulting in robust controllers. Finally, our framework aims to remain general and be compatible with multiple control strategies such as velocity, acceleration or torque control. It can also be used to train both time-dependent [1, 5] or time-independent policies [18].

Python/TensorFlow codes related to this paper can be cloned from the following repository: https://github.com/emmanuelpignat/tf_robot_learning.

2 Generative adversarial training for product of policies

In the generative adversarial framework [14], a generator $G(\boldsymbol{z}; \boldsymbol{\theta})$ is trained to transform input noise $p_z(\boldsymbol{z})$ into samples that look like the data distribution. To do this, a discriminator is trained in parallel to output the probability that a sample comes from the data rather than from the generator. On its side, the generator has to maximize the probability to mislead the discriminator. The generator and discriminator are typically neural networks trained with stochastic gradient descent (SGD). At each step of the training, the discriminator is optimized for a few steps of SGD and then one step is done for the generator.

2.1 State-space generator

In order to generate trajectories, the considered generator is a state-space model defined by several components. We assume that we have access to a stochastic dynamic model of the robot $p(\boldsymbol{\xi}_{t+1}|\boldsymbol{\xi}_t, \boldsymbol{u}_t, \boldsymbol{\theta}_f)$ where $\boldsymbol{\xi}_t$ is the state at time t, $\boldsymbol{u}_t$ the control command and $\boldsymbol{\theta}_f$ the parameters of the dynamics model. If the robot is controlled with inverse dynamics, this model can be a simple integrator, but more complex models such as neural networks can be considered. If the dynamics model is not known or is uncertain, a distribution of parameters $p(\boldsymbol{\theta}_f)$ can be defined, under which the expected objective will be optimized, as we will see Sec. 3. The state of the system being sometimes not directly observed, an additional component that needs to be defined is a stochastic observation model $p(\boldsymbol{y}_t|\boldsymbol{\xi}_t)$ where $\boldsymbol{y}_t$ is an observation. Control commands are computed given this observation by a stochastic policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{y}_t)$ where $\boldsymbol{\theta}$ are the parameters of the policy. A distribution of trajectories $\boldsymbol{\tau} = \{\boldsymbol{y}_1, \boldsymbol{u}_1, ..., \boldsymbol{y}_T, \boldsymbol{u}_T\}$ is then defined as a state-space model with

$$p(\boldsymbol{\tau}|\boldsymbol{\theta}_f, \boldsymbol{\theta}) = p(\boldsymbol{\xi}_1) \prod_{t=1}^{T} p(\boldsymbol{\xi}_{t+1}|\boldsymbol{\xi}_t, \boldsymbol{u}_t, \boldsymbol{\theta}_f)\, \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{y}_t)\, p(\boldsymbol{y}_t|\boldsymbol{\xi}_t). \tag{1}$$

An evident way to learn the MPs is to compute maximum likelihood estimation of $\boldsymbol{\theta}$ given a set of demonstrated trajectories. However, computing or maximizing this likelihood requires approximations or restricting assumptions [19, 20, 2]. In GANs, the likelihood is not modelled explicitly. It is just required to be able to draw samples from this density. Full sequences can be generated by forward sampling, by sampling each model after the other according to (1). Thus, great flexibility is allowed for setting the dynamics, policy and observation models; they only have to be simple to sample from. Additionally, this process should be differentiable with respect to their respective parameters (e.g. $\boldsymbol{\theta}$ and $\boldsymbol{y}$ for the policy model) using the reparametrization trick [21]. If the observation model is not bijective, it might be impossible to retrieve the distribution of initial states of the demonstrations $p(\boldsymbol{\xi}_1)$. In this case, this distribution can be parametrized and optimized as well. For simplicity of the notation, an observable system with $\boldsymbol{\xi}_t = \boldsymbol{y}_t$ will be used for the derivations in the rest of the paper, without loss of generality.

2.1.1 Adaptation with products of Gaussian policies (PoGP)

While the $d_{\boldsymbol{\xi}}$-dimensional state $\boldsymbol{\xi}$ of a robotic manipulator is defined by its joint angles (and possibly velocities), movements are often best explained under several task spaces. Each task space P is associated with a task map, which is a non-linear function $\mathcal{T}_{\boldsymbol{\xi},p} : \mathbb{R}^{d_{\boldsymbol{\xi}}} \to \mathbb{R}^{k_{\boldsymbol{\xi}}^p}$. Accordingly, a set of linear functions maps control commands $\boldsymbol{u}$ (joint velocities or acceleration) to their value in the different task spaces $\mathcal{T}_{\boldsymbol{u},p} : \mathbb{R}^{d_u} \to \mathbb{R}^{k_{\boldsymbol{u}}^p}$. These transformations can be parametrized by the poses of an external object (which we will drop in the notation for simplicity) or by time, which we will denote with the superscript t. We propose to define a stochastic Gaussian policy in each of these task spaces as

$$\mathcal{T}_{\boldsymbol{u},p}^t(\boldsymbol{u}_t) \sim \mathcal{N}\Big(\boldsymbol{\mu}_p^t\big(\mathcal{T}_{\boldsymbol{\xi},p}^t(\boldsymbol{\xi}_t)\big), \boldsymbol{\Sigma}_p^t\big(\mathcal{T}_{\boldsymbol{\xi},p}^t(\boldsymbol{\xi}_t)\big)\Big). \tag{2}$$

In the most general case, $\boldsymbol{\mu}_p^t(\cdot)$ and $\boldsymbol{\Sigma}_p^t(\cdot)$ can be neural networks. In simpler cases, the policies can be proportional-derivative controllers with a constant covariance. In Appendix A, different parameterizations of the policy are given. The proposed overall policy is the fusion of the policies in the different task spaces, given as a product of linearly transformed Gaussians

$$\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{\xi}_t) \propto \prod_{p=1}^{P} \mathcal{N}\Big(\mathcal{T}_{\boldsymbol{u},p}^t(\boldsymbol{u}_t)\Big|\boldsymbol{\mu}_p^t\big(\mathcal{T}_{\boldsymbol{\xi},p}^t(\boldsymbol{\xi}_t)\big), \boldsymbol{\Sigma}_p^t\big(\mathcal{T}_{\boldsymbol{\xi},p}^t(\boldsymbol{\xi}_t)\big)\Big). \tag{3}$$

This product has a closed form expression as a Gaussian and can be sampled directly. Many works [5, 6, 22] have a final controller of this form, but it is computed by using expert policies that have been learned independently.

2.2 Including LfD generative models as close-to-optimal discriminators

Besides training a discriminator as a neural network $D(\boldsymbol{\tau})$, we propose to include standard generative models used in LfD. They are usually trained in closed form or with very efficient procedures like EM. This addition is motivated by a dramatically increased stability and speed of the training procedure. We propose to include a second discriminator $D_q(\boldsymbol{\tau})$ which is multiplied to the original one. This discriminator consists of two approximate distributions q_{samples} and q_{data} learned with standard LfD generative models as [1, 5, 4]. In this work, this discriminator is called a *close-to-optimal discriminator* with

$$D_q(\boldsymbol{\tau}) = \frac{q_{\text{data}}(\boldsymbol{\tau})}{q_{\text{data}}(\boldsymbol{\tau}) + q_{\text{samples}}(\boldsymbol{\tau})}. \tag{4}$$

In an optimal discriminator, q_{data} and q_{samples} would be the exact distributions, not the approximate ones which are improperly normalized. But if we were able to model explicitly this likelihood, the generative adversarial approach would not be needed. The class of distributions q we propose to use typically drops some dependencies or do not integrate to one on the space of trajectories. Another formulation of this problem is that the system is underactuated[2] [23]. Directly used as generative models, where the dropped dependencies are only restored at the synthesis phase [4, 24], these models induce distortions, as discussed in [10]. These distortions are extensively reduced if these models are used as classifiers in the context of generative adversarial learning; both the samples from the generator and the dataset are compared under the same approximations while the feasibility and dependencies are ensured by the generator (1).

We propose to train the approximate distribution q_{data} once at the beginning of the learning process. The approximate distribution of samples q_{samples} is updated with maximum likelihood before each

[2]All the trajectories of the distribution are not feasible.

step of gradient descent of the policy parameters, see Alg. 1. As it might be costly to generate many samples from the generator at each iteration, q_{samples} can be learned incrementally with stochastic updates of maximum likelihood. Such updates can be derived for expectation maximization (EM), closed-form maximum likelihood (e.g. Gaussian distribution) or variational inference [25] (in the case where q_{data} and q_{samples} are Bayesian models whose posterior distribution is estimated).

Many possibilities are offered for choosing the family of approximate distributions q. A very simple choice, if the trajectories are all aligned in time, is to use a factorized Gaussian distribution as

$$q(\boldsymbol{\tau}) = \prod_{t=1}^{T} \mathcal{N}\Big(\begin{bmatrix} \boldsymbol{\xi}_t \\ \boldsymbol{u}_t \end{bmatrix} \Big| \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t \Big). \tag{5}$$

Matching factorized Gaussian distributions is also done in [2] in a similar context. However, q is represented explicitly by using Gaussian process dynamics models and moment matching approximations [20]. If the trajectories have particular correlations across time (that are not due to the dynamics), probabilistic movements primitives (ProMP)[1] can be used instead. In order to provide adaptation to parametrized task spaces, the discriminator can additionally compare the trajectories in these task spaces as

$$q(\boldsymbol{\tau}) = \prod_{t=0}^{T} \left(\mathcal{N}\Big(\begin{bmatrix} \boldsymbol{\xi}_t \\ \boldsymbol{u}_t \end{bmatrix} \Big| \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t \Big) \prod_{p=1}^{P} \mathcal{N}\Big(\begin{bmatrix} \mathcal{T}^t_{\boldsymbol{\xi},p}(\boldsymbol{\xi}_t) \\ \mathcal{T}^t_{\boldsymbol{u},p}(\boldsymbol{u}_t) \end{bmatrix} \Big| \boldsymbol{\mu}^p_t, \boldsymbol{\Sigma}^p_t \Big) \right). \tag{6}$$

In this case, even if the approximate distributions q are Gaussians, the discriminator would be able to distinguish between more complex distributions, as the comparison is done under non-linear transformations. In the case where the trajectories cannot be time-aligned or that the targeted distribution is multimodal, more complex models as hidden Markov models [4] or Gaussian mixture models [22] can be used, with the density

$$q(\boldsymbol{\tau}) = \prod_{t=0}^{T} \left(\sum_{k=0}^{K} \pi_k \, \mathcal{N}\Big(\begin{bmatrix} \boldsymbol{\xi}_t \\ \boldsymbol{u}_t \end{bmatrix} \Big| \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \Big) \right). \tag{7}$$

They can be trained very efficiently with EM in a few milliseconds.

Algorithm 1: Robust generative adversarial training of movement primitives

```
1 Compute maximum likelihood of $q_{\text{data}}$ on the $N$ demonstrations $\{\hat{\boldsymbol{\tau}}^{(i)}\}_{i=1}^N$
2 for number of training iterations do
3     for $L$ dynamic models $\{\boldsymbol{\theta}_f{}^{(j)}\}_{j=1}^L$ do
4         Sample from (1) $M$ trajectories $\{\boldsymbol{\tau}^{(i,j)}\}_{i=1}^M$
5         Apply (stochastic) maximum likelihood update on $q_{\text{samples}}{}^{(j)}$ given $\{\boldsymbol{\tau}^{(i,j)}\}_{i=1}^M$
6     end
7     Update global policy parameters $\boldsymbol{\theta}$ by descending the stochastic gradient:
          $\nabla_{\boldsymbol{\theta}} \frac{1}{ML} \sum_{j=1}^{L} \sum_{i=1}^{M} \Big( -\log\big(q_{\text{data}}(\boldsymbol{\tau}^{(i,j)})\big) + \log\big(q_{\text{data}}(\boldsymbol{\tau}^{(i,j)}) + q_{\text{samples}}{}^{(j)}(\boldsymbol{\tau}^{(i,j)})\big) - \log\big(D(\boldsymbol{\tau}^{(i,j)})\big)\Big)$   (8)
8 end
```

By using Gaussian models, which have quadratic log-likelihood, the gradients are well-behaved, leading to fast convergence. At initialization, our generator samples trajectories $\boldsymbol{\tau}^{(i)}$ very far from q_{data} but close from q_{samples}. The term $\frac{1}{M}\sum_{i=1}^{M} -\log\big(q_{\text{data}}(\boldsymbol{\tau}^{(i)})\big)$ in Alg. 1 would dominate the gradient of the cost, which would be close to quadratic. For practical and stability reasons, we propose to train the policy first by considering only the additional classifier using approximate distributions. Then, the neural network classifier is only used for additional refinements and with smaller learning rates. In this case, the neural network is just helping the additional classifier to distinguish features that are not encoded in the approximate distributions q.

3 Robustness and unknown dynamics

In this section, we propose a strategy to learn robust policies in case of changing or unknown dynamics. So far, we have considered fixed parameters $\boldsymbol{\theta}_f$ of the stochastic dynamic system. It can result in a poor matching of the trajectory distribution in the case where the model of dynamics $\boldsymbol{\theta}_f$ does not match the real system. In the worst case, the distributions can completely diverge and executing the policy can be dangerous. In other cases, the problems can be less disastrous and more subtle. For example, if the model overestimates the stochasticity of the system, the rollouts on the real system would have lower variance than the demonstrations. In this case, the system will rely too much on the stochasticity of the environment to create variability. A robust policy has to match the distribution of trajectories for a distribution of parameters $p(\boldsymbol{\theta}_f)$. This distribution can be either a hand-tuned prior distribution, a posterior distribution if the dynamics are learned with Bayesian methods or a set of parameters $\{\boldsymbol{\theta}_f{}^{(j)}\}_{j=1}^{L}$ if they are learned with ensemble methods.

We propose to condition the discriminator on $\boldsymbol{\theta}_f$, which means that this value should be fed to it together with the samples. As the true system is not known when executing the policy, this latter should not depend on the parameters of the dynamics. Giving access to the model parameters on which are generated the samples to the discriminator only forces the policy to match the distribution of data under a distribution of dynamic parameters, ensuring robustness.

In order to use the additional discriminators $D_q(\boldsymbol{\tau})$ proposed in Section 2.2, two alternatives are possible. The choice mainly depends on the trade-off between robustness and computation time. In both cases, multiple approximate models $\{q_{\text{samples}}{}^{(j)}\}_{j=1}^{L}$ are learned on a batch of dynamic parameters $\{\boldsymbol{\theta}_f{}^{(j)}\}_{j=1}^{L}$. In the case of privileging robustness, the L dynamic parameters are drawn from their distribution before each iteration of gradient descent. In this case, enough samples should be drawn from (1) in order to compute the maximum likelihood of the approximate distribution $q_{\text{samples}}{}^{(j)}$. The stochastic updates are not allowed as the L model from the previous iterations do not correspond anymore. When it is too costly to sample enough trajectories to perform complete maximum likelihood of q, the L models can be changed only after a given number of iterations or even kept fixed throughout the learning process. This solution is also natural if the parameters are learned by an ensemble method. The procedure is more formally presented in Alg. 1.

The parameters of the system $\boldsymbol{\theta}_f$ can be also learned or refined. For model-based policy search (from which our approach is a particular case), a key requirement of the dynamic model is its ability to produce good long-term predictions. Many approaches optimize a one-step-ahead model, for example by maximizing the likelihood of $p(\boldsymbol{\xi}_{t+1}|\boldsymbol{\xi}_t, \boldsymbol{u}_t)$. However, due to modelling errors, false assumptions on the model and noisy or partial observation model, this approach tends to produce brittle predictions which diverge quickly from the real system [26]. Robust approaches such as [19] optimize the likelihood of full sequences of observations $p(\boldsymbol{y}_1, \dots, \boldsymbol{y}_T)$ marginalized on the sequence of latent states $\{\boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_T\}$. Our approach optimizes the same objective but in the generative adversarial framework, which does not require to model explicitly the marginal distribution of observation. It makes very little assumptions on the dynamic, observation and policy models at the expense of a higher computational cost.

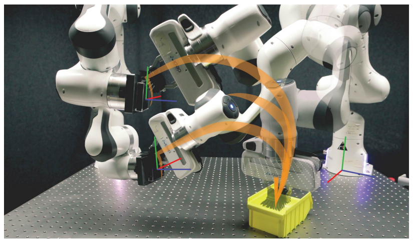
The approach to refine the dynamic parameters is presented in the case of the close-to-optimal discriminator $D_q(\boldsymbol{\tau})$. The proposed process alternates between learning parameters of the policy using Alg. 1 and executing this policy on the real system to update the dynamic model. To do so, an additional approximate distribution $q_{\boldsymbol{\theta},\text{data}}$ is introduced. It models the distribution of trajectories executed on the real system with the inferred policy parameters $\boldsymbol{\theta}$ of the previous step. The distribution of trajectories q_{samples} sampled with the inferred policy and model of the dynamics is optimized to match this new distribution $q_{\boldsymbol{\theta},\text{data}}$. This time, the gradient is computed with respect to the parameters of the dynamic model $\boldsymbol{\theta}_f$. For increased robustness, it is better to keep multiple dynamic parameters $\{\boldsymbol{\theta}_f{}^{(j)}\}_{j=1}^{L}$ and train them in parallel as an ensemble method, see Alg. 2. For example, if the inertia of the robot is not known, the multiple initial dynamic parameters could reflect this. The policy training at the first iteration (before executing on the real system) would be more conservative (high feedback terms) to accommodate this uncertainty. In the case where only one dynamic parameter was used in the policy optimization, the execution on the robot can be disastrous (for example if the inertia matrix was overestimated).

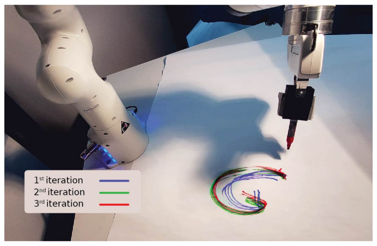
4 Experiments

We present two illustrative experiments performed on a 7-DoF Panda robot. Other experiments with quantitative evaluations are performed on synthetic data and presented in Appendix B.

Algorithm 2: Refining dynamic models with an ensemble method

```
Compute maximum likelihood of $q_{\text{data}}$ on the $N$ demonstrations $\{\hat{\boldsymbol{\tau}}^{(i)}\}_{i=1}^{N}$
for number of real-system iterations do
    Start with $L$ initial guesses $\{\boldsymbol{\theta}_f{}^{(j)}\}_{j=1}^{L}$ of system dynamics
    Update policy with Algorithm 1
    Sample $n$ trajectories $\{\hat{\boldsymbol{\tau}}_{\boldsymbol{\theta}}^{(i)}\}_{i=1}^{N}$ on the real system given current policy parameters $\boldsymbol{\theta}$
    Compute maximum likelihood of the distribution of new trajectories $q_{\boldsymbol{\theta},\text{data}}$
    for number of training iterations do
        for $L$ dynamic models $\{\boldsymbol{\theta}_f{}^{(j)}\}_{j=1}^{L}$ do
            Sample from (1) $M$ trajectories $\{\boldsymbol{\tau}^{(i,j)}\}_{i=1}^{M}$
            Apply (stochastic) maximum likelihood update on $q_{\text{samples}}{}^{(j)}$ given $\{\boldsymbol{\tau}^{(i,j)}\}_{i=1}^{M}$
            Update dynamic model parameters $\boldsymbol{\theta}_f{}^{(j)}$ using the stochastic gradient:
                $$\nabla_{\boldsymbol{\theta}_f{}^{(j)}} \frac{1}{M}\sum_{i=1}^{M} -\log\left(q_{\boldsymbol{\theta},\text{data}}(\boldsymbol{\tau}^{(i,j)})\right) + \log\left(q_{\boldsymbol{\theta},\text{data}}(\boldsymbol{\tau}^{(i,j)}) + q_{\text{samples}}{}^{(j)}(\boldsymbol{\tau}^{(i,j)})\right) \quad (9)$$
        end
    end
end
```


(a) Painting task with adaptation to varying poses.



(b) Drawing task in varying environments.

Figure 1: Two illustrative tasks are performed on the robot to demonstrate the adaptation and robustness of the approach. A 7-DoF Panda robot is used in the experiments, controlled with acceleration commands in (a), and with torque commands in (b).

4.1 Acceleration control with adaptation

In this experiment, the robot has to paint a box held by another robot. It needs to dip the brush in a paint container that is always at the same place and then wipe the box whose orientation and position can vary, see Fig. 1a. The dataset consists of $N = 7$ time-aligned demonstrations of 4.5 s with a discretization of time $dt = 0.02$ s. In each demonstration, the box has a different pose. We consider that the control commands are the joint accelerations $\ddot{\boldsymbol{q}} \in \mathbb{R}^7$ and the state $\boldsymbol{\xi} \in \mathbb{R}^{14}$ consists of joint angles and velocities of the robot holding the brush. In this configuration, the dynamic model is thus given as a double integrator.

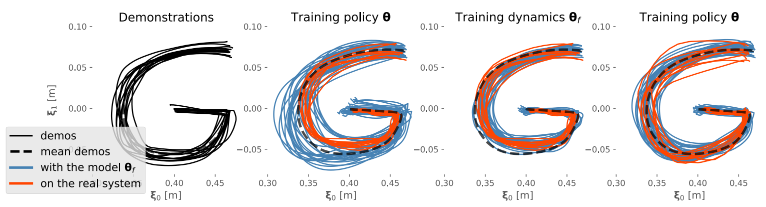
In order to provide adaptation, the policy and discriminator are defined in joint space and two task spaces. The first task space is the position and orientation of the end-effector in a fixed coordinate system and the second is in a coordinate system attached to the box to paint. In each of these three spaces, a Gaussian feedback controller with time-varying gains, feed-forward terms and covariances are defined. More details are given in Appendix A, Equation (15). The discriminator consists of a factorized Gaussian distribution in each task space, as in (6).

As evaluation, we compare the adaptation capabilities with a ProMP conditioned on the 6-DoF pose of the box. As metric, we compute the Bhattacharyya distance[3] (BD) for the distribution of final position and orientation in the coordinate system of the box between the demonstrations and the reproductions. These final poses are shown in Fig. 1a. Results are reported in Table 1. They first are performed on the 7 different contexts of the demonstrations. In this case, ProMP conditioning gives better results. Generalization is then tested by sampling 20 contexts from the Gaussian distribution of poses in the demonstrations. To analyze the extrapolation capabilities, the standard deviation is

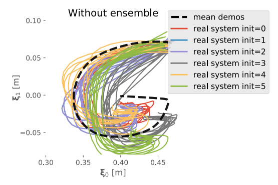
[3]This distance is computed by approximating the final distribution with a Gaussian on 10 trajectory samples.

Table 1: Bhattacharyya distance as quantitative evaluation for the painting task.

	Training	**Testing** $\sigma_c = 1$	**Testing** $\sigma_c = \sqrt{2}$	**Testing** $\sigma_c = 2$
ProMP conditioning [1]	$\mathbf{0.35 \pm 0.13}$	1.72 ± 2.29	8.18 ± 15.86	13.21 ± 22.66
GAMP	0.60 ± 0.23	$\mathbf{0.79 \pm 0.81}$	$\mathbf{1.20 \pm 0.84}$	$\mathbf{7.02 \pm 17.78}$



(a) Iterations of Alg. 2 using an ensemble method to learn the dynamics.



(b) First execution without the ensemble method.

Figure 2: Learning to reproduce the distribution of "G" letter on a 7-DoF Panda robot.

multiplied by $\sigma_c \in \{1, \sqrt{2}, 2\}$. In every case unseen in the demonstrations, the product of policies generalizes better.

4.2 Force control and dynamic model learning

In this experiment, we reproduce 2D handwritten letters from [4] in different environments. For each letter, $N = 13$ time-aligned demonstrations of $T = 200$ timesteps are given. With a discretization of time $dt = 0.01$ s, the trajectories last 2 s. The policy learned is then run at 1000 hz on the robot. A third dimension is added to the letters as a fixed height. We alternate between optimizing the policy and refining the model of the system, as proposed in Alg. 2. We consider that the control commands are the forces $\mathbf{f} \in \mathbb{R}^3$ applied at the end-effector as $\boldsymbol{\tau} = \boldsymbol{J}^\top \mathbf{f}$. The state $\boldsymbol{\xi} \in \mathbb{R}^6$ is the position and velocity of the end-effector. In this experimental setup, the configuration of the robot is considered as a hidden variable that influences the dynamics. This uncertainty has to be learned by the identification of the dynamic parameters and the policy robust to the unknown configuration.

A time-dependent feedback controller is used as in the first experiment. The dynamics are learned by an ensemble method. We consider that the system is a mass of 3 kg on which a non-linear state-dependent perturbation is added. This non-linear term is modeled as a MLP with two hidden layers of 20 units, $tanh$ activation and the last layer linear. At initialization, the neural networks generate perturbations of a standard deviation of 5 N. This initialization is important: if the true system is in the distribution of systems defined by the initialization of the L models, then the first policy executed on the robot will be already quite good. We demonstrate this by performing the same task with a unique neural network instead of an ensemble.

After initialization of the dynamic parameters, a robust policy is learned for 10 s. This policy is run on the robot for $M = 10$ times, by starting at a random initial state of the demonstrations, and with a random configuration. With the initial guesses about the system, the first computed policy already leads to a very similar distribution, see Fig. 2a (second column). The L dynamic parameters are optimized in parallel for 10 s given these new trajectories. The trajectories of the generator now match the trajectories on the true system (third column). The whole process can be repeated until convergence. In this experiment, the policy has been updated only once more for $10s$ and tested on the robot with good results (fourth column).

As a comparison, Fig. 2b shows the execution of the first policy on the system when no ensemble method are used. Several sets of trajectories are displayed corresponding to different initializations of $\boldsymbol{\theta}_f$. In this case, the trajectories are worse than the ensemble method because the epistemic uncertainty is not taken into account, which results in an overconfidence on the dynamic model. As a comparison, the first policy computed using the ensemble method (Fig. 2a) has higher feedback gains, resulting in a lower sensitivity to uncertainties.

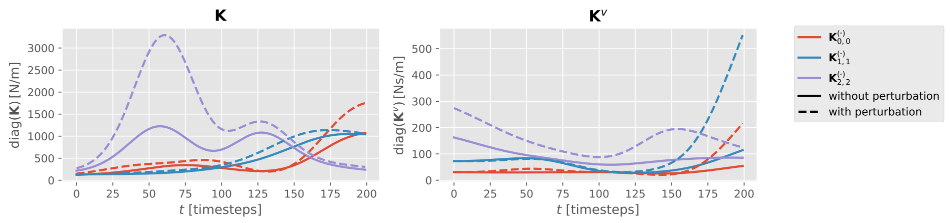


Figure 3: Increase of feedback gains resulting from the identification of external perturbations. *Left:* Diagonal values of the time-dependent proportional gains $\boldsymbol{K}(t) \in \mathbb{R}^{3\times3}$. *Right:* Diagonal values of the derivative gains $\boldsymbol{K}^v(t) \in \mathbb{R}^{3\times3}$.

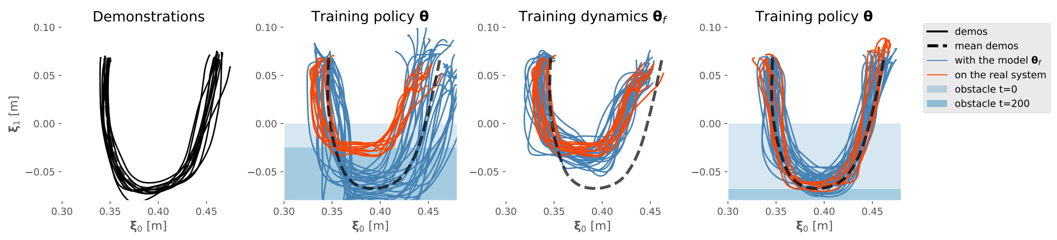


Figure 4: Iterations of Alg. 2 for reproducing a distribution of "U" shaped trajectories with an obstacle that should be pushed.

These first sets of trajectories were produced without any other perturbations than the unknown configuration. To assess for the generality of the method, we tested the process in three different environments. In the first, a user applies short (around 0.2 s) perturbations of around 10 N all along the trajectories and in every direction. After an update of the dynamic model, a higher stochasticity of the environment is inferred. The following update of the policy results in higher feedback terms (see Fig. 3).

In a second environment, we tested if the approach is able to learn that an obstacle is on its path, which should be pushed. This time, the letter "U" was chosen and a moving plastic block of around 1 kg put on the table to block the lower part of the letter. Iterations of Alg. 2 are shown for this environment in Fig. 4. The first trajectories executed on the robot are truncated (second column). The friction between the end-effector of the robot and the obstacle also prevents motion along $\boldsymbol{\xi}_0$. After updating the dynamics model with $M = 10$ rollouts, the prediction of the generator matches the real system (third column). The following update of the policy leads to a much better reproduction of the distribution (fourth column).

In the third case, the robot needs to draw the letter on a paper. A pen was placed in the gripper of the robot and the end-effector redefined as the tip of the pen. The policy was constrained to apply a constant force of 8 N on the paper, as this information was not in the dataset. The contact of the table was explicitly modeled in the dynamics model as a spring-damper system with high gains. The ensemble method still had to learn the additional friction induced by the tip of the pen on the paper. This dynamics was harder to train and the system needed three iterations of the whole process instead of one in the previous experiments. The trajectories of these iterations are shown in Fig. 1b.

5 Conclusion

The generative adversarial framework is promising for learning movement primitives. It can bring together numerous classical techniques from LfD with the computation power and flexibility of modern machine learning architecture [27]. The approach is easy to be adapted to a wide range of problems. A practitioner vaguely familiar with machine learning is only required to define a function for the dynamic system, one for the policy and possibly multiple relevant task spaces. Future work will focus on learning more complex policies and dynamics models, also from raw pixel observations. More efficient model-based policy search methods should also be incorporated in the framework, to cope with longer horizon problems.

Acknowledgments

The research leading to these results has received funding from the European Commission's Horizon 2020 Programme through the MEMMO Project (Memory of Motion, http://www.memmo-project.eu/, grant agreement 780684) and CoLLaboratE project (https://collaborate-project.eu/, grant agreement 820767).

References

[1] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann. Probabilistic movement primitives. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2616–2624, 2013.

[2] P. Englert, A. Paraschos, M. P. Deisenroth, and J. Peters. Probabilistic model-based imitation learning. *Adaptive Behavior*, 21(5):388–403, 2013.

[3] S. Levine and V. Koltun. Guided policy search. In *Proc. Intl Conf. on Machine Learning (ICML)*, pages 1–9, 2013.

[4] S. Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics*, 9(1):1–29, 2016.

[5] S. Calinon and A. Billard. Statistical learning by imitation of competing constraints in joint space and task space. *Advanced Robotics*, 23(15):2059–2076, 2009.

[6] A. Paraschos, R. Lioutikov, J. Peters, and G. Neumann. Probabilistic prioritization of movement primitives. *IEEE Robotics and Automation Letters*, 2(4):2294–2301, 2017.

[7] S. Niekum, S. Osentoski, G. Konidaris, S. Chitta, B. Marthi, and A. G. Barto. Learning grounded finite-state representations from unstructured demonstrations. *The International Journal of Robotics Research*, 34(2):131–157, 2015.

[8] M. Mühlig, M. Gienger, J. J. Steil, and C. Goerick. Automatic selection of task spaces for imitation learning. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pages 4996–5002, 2009.

[9] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[10] H. Zen, M. J. Gales, Y. Nankaku, and K. Tokuda. Product of experts for statistical parametric speech synthesis. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(3): 794–805, 2012.

[11] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[12] C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *Proc. Intl Conf. on Machine Learning (ICML)*, pages 49–58, 2016.

[13] M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal. Learning objective functions for manipulation. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 1331–1336, 2013.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014.

[15] C. Finn, P. Christiano, P. Abbeel, and S. Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *NeurIPS Workshop on Adversarial Training*, 2016.

[16] K. Hausman, Y. Chebotar, S. Schaal, G. Sukhatme, and J. J. Lim. Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1235–1245, 2017.

[17] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4565–4573, 2016.

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

[19] A. Doerr, C. Daniel, M. Schiegg, N.-T. Duy, S. Schaal, M. Toussaint, and T. Sebastian. Probabilistic recurrent state-space models. In *Proc. Intl Conf. on Machine Learning (ICML)*, pages 1280–1289, 2018.

[20] M. Deisenroth and C. E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proc. Intl Conf. on Machine Learning (ICML)*, pages 465–472, 2011.

[21] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *Proc. Intl Conf. on Learning Representation (ICLR)*, 2013.

[22] E. Pignat and S. Calinon. Bayesian Gaussian mixture model for robotic policy imitation. *IEEE Robotics and Automation Letters (RA-L)*, 4(4):4452–4458, 2019. doi:10.1109/LRA.2019.2932610.

[23] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics*, 7(1-2):1–179, 2018.

[24] A. Paraschos. *Robot Skill Representation, Learning and Control with Probabilistic Movement Primitives*. PhD thesis, Technische Universität Darmstadt, 2017.

[25] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[26] S. A. Billings. Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains. *John Wiley and Sons*, 2013.

[27] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th Symposium on Operating Systems Design and Implementation*, pages 265–283, 2016.

[28] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal. Operational space control: A theoretical and empirical comparison. *The International Journal of Robotics Research*, 27(6): 737–757, 2008.

[29] M. Bohner and N. Wintz. The linear quadratic tracker on time scales. *International Journal of Dynamical Systems and Differential Equations*, 3(4):423–447, 2011.

[30] H. Zen, K. Tokuda, and T. Kitamura. Reformulating the HMM as a trajectory model by imposing explicit relationships between static and dynamic feature vector sequences. *Computer Speech & Language*, 21(1):153–173, 2007.

[31] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proc. Intl Conf. on Artificial Intelligence and Statistics (AISTATS)*, pages 627–635, 2011.

A Robotic policies/controllers

In this appendix, we propose several convenient parameterizations of policies that can be used in our framework. As proposed in Sec. 2.1.1, the policies used in this work are defined in P task spaces $\mathcal{T}_p : \mathbb{R}^d \to \mathbb{R}^{k_p}$. We denote $\boldsymbol{x}_p = \mathcal{T}_p(\boldsymbol{q})$ the value in task space p and $\boldsymbol{J}_p = \partial \mathcal{T}_p / \partial \boldsymbol{q}$ its Jacobian. The velocity $\dot{\boldsymbol{x}}_p$ and acceleration $\ddot{\boldsymbol{x}}_p$ in the task space are

$$\dot{\boldsymbol{x}}_p = \boldsymbol{J}_p(\boldsymbol{q})\dot{\boldsymbol{q}}, \tag{10}$$

$$\ddot{\boldsymbol{x}}_p = \boldsymbol{J}_p(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_p(\boldsymbol{q})\dot{\boldsymbol{q}} \approx \boldsymbol{J}_p(\boldsymbol{q})\ddot{\boldsymbol{q}}. \tag{11}$$

Table 2: Equivalences between abstract state $\boldsymbol{\xi}$, control command $\boldsymbol{u}$, task-spaces transform and robotic variables.

Control strategy	Velocity	Acceleration	Force
State $\boldsymbol{\xi}$	$\boldsymbol{q}$	$\begin{bmatrix}\boldsymbol{q}\\ \dot{\boldsymbol{q}}\end{bmatrix}$	$\begin{bmatrix}\boldsymbol{q}\\ \dot{\boldsymbol{q}}\end{bmatrix}$
Control command $\boldsymbol{u}$	$\hat{\dot{\boldsymbol{q}}}$	$\hat{\ddot{\boldsymbol{q}}}$	$\boldsymbol{\tau}$
Transform $\mathcal{T}_{\boldsymbol{\xi},p}$	$\mathcal{T}_p(\boldsymbol{q})$	$\begin{bmatrix}\mathcal{T}_p(\boldsymbol{q})\\ \boldsymbol{J}_p(\boldsymbol{q})\dot{\boldsymbol{q}}\end{bmatrix}$	$\begin{bmatrix}\mathcal{T}_p(\boldsymbol{q})\\ \boldsymbol{J}_p(\boldsymbol{q})\dot{\boldsymbol{q}}\end{bmatrix}$
Transform $\mathcal{T}_{\boldsymbol{u},p}$	$\boldsymbol{J}_p(\boldsymbol{q})\hat{\dot{\boldsymbol{q}}}$	$\boldsymbol{J}_p(\boldsymbol{q})\hat{\ddot{\boldsymbol{q}}}$	$\boldsymbol{J}_p^{\top\dagger}(\boldsymbol{q})\boldsymbol{\tau}$

The relation between the joint torque $\boldsymbol{\tau}$ and the generalized force is

$$\boldsymbol{J}_p^{\top}(\boldsymbol{q})\,\mathbf{f}_p = \boldsymbol{\tau}. \tag{12}$$

These relations are used to define an equivalence between variables used in the above and the different control strategies. The equivalences are reported in Table 2 for different control strategies. For velocity and acceleration control, $\hat{\dot{\boldsymbol{q}}}$ and $\hat{\ddot{\boldsymbol{q}}}$ are reference values that are tracked by lower-level controller, such as inverse dynamics [28].

These relations do not need to be exact. They are just parameterizations of the policy which give a better structure to the problem to facilitate the training phase and increase generalization capabilities. Simplifications, such as dropping $\dot{\boldsymbol{J}}_p(\boldsymbol{q})$ for acceleration control, can be done to speed up the computation of the stochastic gradient while training.

The policies can be parametrized in different ways. For time-dependent policies, a solution is a feedback controller with time-varying gains and feed-forward terms. These controllers are very usual in LfD [1, 4] and are also solutions of linear-quadratic tracking problems [29]. They can be used both for velocity control

$$\boldsymbol{\mu}_p^t(\mathcal{T}_{\boldsymbol{\xi},p}^t(\boldsymbol{\xi})) = -\boldsymbol{K}_p(t)\mathcal{T}_p^t(\boldsymbol{q}) + \boldsymbol{d}_p(t), \tag{13}$$

$$\boldsymbol{\Sigma}_p^t(\mathcal{T}_{\boldsymbol{\xi},p}^t(\boldsymbol{\xi})) = \hat{\boldsymbol{\Sigma}}_p(t), \tag{14}$$

or acceleration and force with

$$\boldsymbol{\mu}_p^t(\mathcal{T}_{\boldsymbol{\xi},p}^t(\boldsymbol{\xi})) = -\boldsymbol{K}_p(t)\mathcal{T}_p^t(\boldsymbol{q}) - \boldsymbol{K}_p^v(t)\boldsymbol{J}_p(\boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{d}_p(t). \tag{15}$$

Continuous values for the parameters depending on time t can be induced by linear basis functions or simple multilayer perceptrons (MLP). Gains $\boldsymbol{K}$ can be parametrized in several restrictive ways depending on the assumptions on the system. It can help at stabilizing and speeding up the training phase as well as at providing more safety on the robot.

Time-independent policy can be defined with MLPs that output the parameters of the Gaussian policy

$$\boldsymbol{\mu}_p^t(\mathcal{T}_{\boldsymbol{\xi},p}^t(\boldsymbol{\xi})) = \mathrm{F}_{\boldsymbol{\mu}}\big(\mathcal{T}_p^t(\boldsymbol{q})\big), \quad \boldsymbol{\Sigma}_p^t(\mathcal{T}_{\boldsymbol{\xi},p}^t(\boldsymbol{\xi})) = \mathrm{F}_{\boldsymbol{\Sigma}}\big(\mathcal{T}_p^t(\boldsymbol{q})\big). \tag{16}$$

Covariance matrices can be parametrized by their Cholesky decomposition or using the matrix exponential of another symmetric matrix. Time-independent policies have also been modeled by conditioning in Gaussian mixture models [18, 22]. This latter approach is extremely fast to train from data but its gradient is not well-behaved for optimization.

B Additional experiments with synthetic data

In this appendix, we propose additional experiments with synthetic data and more exhaustive quantitative evaluations. In the first experiment, we compare the matching of distributions under simulated perturbations. In the second experiment, we learn a context-dependent time-independent policy and test its robustness and generalization capabilities.

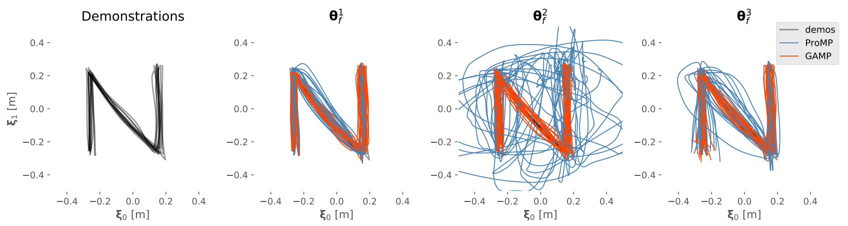


Figure 5: Demonstrations and reproductions using ProMP and GAMP for different stochasticity of the environment. The same controllers are used in the 3 situations.

B.1 Time-dependent policy

In this first experiment with synthetic systems, we consider a simulated 2D unit mass system with a discretization of time $dt = 0.01$ s. The state $\boldsymbol{\xi} \in \mathbb{R}^4$ is composed of its position and velocity, and the control command $\boldsymbol{u} \in \mathbb{R}^2$ the force. The dataset are letters from the alphabet [4]. For each letter, $N = 13$ time-aligned demonstrations of $T = 200$ timesteps are given. Demonstrations are shown in Fig. 5-(*left*) for letter "N". A time-dependent feedback Gaussian policy as in (15) is used. The gain matrices and feed-forward terms are parametrized with time-dependent basis functions. Gains are parametrized in several ways, which has more influence on training time than on the final results. For the evaluations, $\boldsymbol{K}_p(t)$ and $\boldsymbol{K}_p^v(t)$ were chosen as diagonal matrices with positives elements. In this first experiment, the policy is not a POGP as it is defined only in the original state space. The approximate distributions used for the discriminator are factorized Gaussians as in (5). Each letter was trained for 10s. The approach presented in Sec. 3 was used on a distribution of dynamic parameters $p(\boldsymbol{\theta}_f)$ which include different values of Gaussian perturbations in force and on the initial state.

We compare our approach with ProMP [1] and hybrid approaches that learn a time-dependent distribution of states with either Gaussian mixture regression (GMR + LQT) [5] or hidden Markov models (HMM + LQT) [4] and use linear quadratic tracker to regenerate continuous trajectories.

Given the controller computed for each model, we evaluated rollouts in 3 situations. In the first case, ${\boldsymbol{\theta}_f}^{(1)}$, the system is deterministic. In the second case, ${\boldsymbol{\theta}_f}^{(2)}$, uncorrelated Gaussian perturbations in force of standard deviation of 10 N are injected. In the third case, ${\boldsymbol{\theta}_f}^{(3)}$ Gaussian noise on the initial position of standard deviation of 0.035 m is applied. Two metrics are used to compare the demonstrations with the synthesized samples. The first one evaluates if the mean motion is well reproduced. For each letter, the mean squared error (MSE) is computed between the mean trajectories (position and velocity only) over the $N = 13$ demonstrations and the mean over 20 samples from the model. The second metric evaluates if the full distribution is well reproduced. The Bhattacharyya distance (BD) is computed over a Gaussian approximation of the distribution of demonstrations and samples.

The results are reported in Table 3 for each case of dynamics, each model and the two metrics. The mean value and standard deviation of these metrics over the whole alphabet is given. Demonstrations and synthesized samples from ProMP and GAMP are shown in Fig. 5 for each stochasticity. ProMP and GAMP have similar results in terms of MSE and BD in case of no stochasticity ${\boldsymbol{\theta}_f}^{(1)}$. The ProMP controller derived in [1] assumes known dynamics and stochasticity in order to match the distribution of demonstrations. Our approach can generate a controller that matches the distribution, for a distribution of stochasticity of the environment. A more robust controller for ProMP can be derived in our framework by using q_{data} and q_{samples} as ProMP distributions. The approach using GMR + LQT is robust to perturbations and performs well in terms of MSE. However, the distribution is not matched very well. The velocity of the rollouts have the same variance as the demonstrations but the positions tend to shrink on the mean trajectory. This is due to the assumption of independence between two consecutive states that are ignored in GMR and HMM and that are later restored with LQT. The proper way to generate the matching distribution would be to use IOC with LQT [15] or trajectory HMM [30].

Table 3: Mean squared error and Bhattacharyya distance between demonstrations and samples of different models. The samples are generated with three different dynamic parameters.

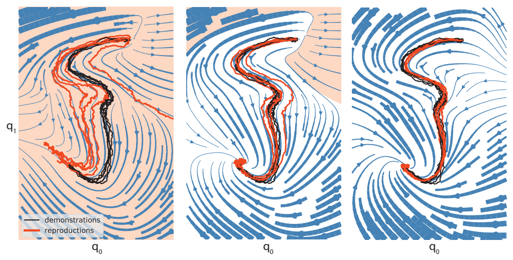
Metrics	**Mean squared error**			**Bhattacharyya distance**		
Environment $\boldsymbol{\theta}_f$	$\boldsymbol{\theta}_f^{(1)}$	$\boldsymbol{\theta}_f^{(2)}$	$\boldsymbol{\theta}_f^{(3)}$	$\boldsymbol{\theta}_f^{(1)}$	$\boldsymbol{\theta}_f^{(2)}$	$\boldsymbol{\theta}_f^{(3)}$
GAMP (ours)	**0.18 ± 0.06**	**0.19 ± 0.08**	**0.23 ± 0.11**	**0.13 ± 0.04**	**0.13 ± 0.04**	**0.11 ± 0.04**
ProMP [1]	0.38 ± 0.27	1.15 ± 0.55	0.60 ± 0.42	0.24 ± 0.10	0.90 ± 0.30	0.29 ± 0.10
GMR + LQT [5]	0.31 ± 0.14	0.34 ± 0.13	0.31 ± 0.13	0.40 ± 0.07	0.30 ± 0.06	0.40 ± 0.07
HMM + LQT [4]	1.58 ± 0.48	1.20 ± 0.47	1.20 ± 0.48	0.76 ± 0.21	0.58 ± 0.21	0.75 ± 0.21

B.2 Time-independent policy in two task spaces

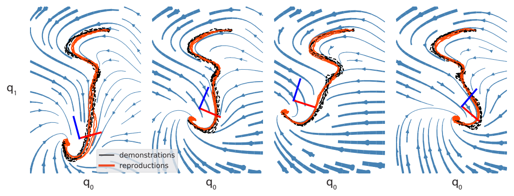
In the second experiment, we consider a time-independent policy that should adapt to a moving object, as shown in Fig. 6b. The system considered is a simple integrator with possible perturbations. The dataset consists of a smooth blend between letters "S" and "J". The letter "J" should move according to an object. Ten different positions and orientations of the object are given, and for each of which $N = 10$ demonstrations of $T = 400$ timesteps are performed. The dataset was randomly split into 5 situations to train and 5 to test the generalization. The policy is the product of two time-independent Gaussian policies given by an MLP as in (16). These two policies are defined on a different task space: the first one on a fixed task space, and the second one, projected in the coordinate system of the moving object. The MLPs have both 2 hidden layers of 150 units with $tanh$ activation and output a state-dependent Gaussian with a full covariance. The covariance is parametrized by its matrix logarithm. Even if the demonstrations are full and aligned, we discard this information for training, and randomly split them in small chunks. The approximate distributions used for the discriminator are Gaussian mixture models with $K = 20$ as in (7) in each task space. Before each update of policy parameters $\boldsymbol{\theta}$, 10 steps of EM are performed with 1000 points each. Every 50 steps of policy parameters update, the mixture models are reinitialized with k-means to avoid local minima. The policy parameters are initialized by maximum likelihood of the policy density on pairs of $\{\boldsymbol{\xi}, \boldsymbol{u}\}$ for 5 s of stochastic gradient descent. This initialization corresponds to a policy imitation objective, which is known to produce brittle policy [31]. Fig. 6a-*(left)* shows the policy after initialization and the dangers of drifting away from the training data. The models are further trained for 15 s in the generative adversarial network. Two alternatives are considered. In the first (GAMP), the system is assumed to be deterministic during training. In the second (GAMP + noise), small perturbations in the initial state of the chunks are simulated. The policy is then trained to look like the demonstrations, even with noise, which results in more robustness. The differences between the policy learned in these two cases are shown in Fig. 6a-*(middle and right)*. We also consider another policy, where the adaptation to the moving object is given by the neural network instead of the usage of multiple task spaces. In this case, the MLP defining the Gaussian policy has an additional input. It is the position and vectorized rotation matrix of the object.

As an evaluation, we produce full rollouts from the initial states of the demonstrations. We compute the mean absolute error (MAE) over the position of the whole rollout and the closest demonstration. The mean value and its standard deviation over the training and testing situations are given in Table 4. The adaptation with the use of task spaces and MLP perform the same on the training set but the former generalizes better. In robotics, many adaptations of movements can be understood easily by projecting them into several coordinate systems. The two approaches can also be combined in the case where the definition of multiple task spaces is not sufficient. The addition of noise in the initial state makes the GAMP more robust. They also generalize better to new situations. When using the imitation cost only, the trajectories diverge, as shown in Fig. 6a-(*left*). In the case of MLP adaptation, they diverge extremely fast, leading to an enormous cost. When using the imitation cost only, the benefits of defining two policies on low-dimensional task space instead of a unique policy with an additional vector of inputs are important. The fusion of two robust policies will tend to be more robust than a policy that can change completely for each new vector defining the situation.

The problem of learning robust policy is very difficult [31], [18]. Our approach gives no guarantees that the system cannot diverge, but the cost of mimicking the distribution of trajectories greatly increases the robustness. By injecting noise and training with stochastic gradient descent for a sufficient amount of time, the system is expected not to diverge. We performed an additional test for showing that this also applies to higher-dimensional systems, given slightly longer optimization. We



(a) Demonstrations and samples with several learning strategies. The area leading to divergence are highlighted. (*left*) The policy is trained only by maximizing its own likelihood, as in policy imitation. (*middle*) The policy is trained with the method presented in this paper, to maximize the likelihood of the trajectories. (*right*) Additional perturbations are added in the initial state of each chunk, leading to a more robust policy.



(b) Demonstrations and samples in several situations. The transformation of the moving object is indicated with a coordinate system. The flow field displays only the mean value of the policy.

Figure 6: Illustration of time-independent policies as flow fields.

Table 4: Mean absolute error (MAE) between the whole rollout and the closest demonstrations for situations in the training and testing set. The red colour indicates huge errors because of divergence.

	Training	**Testing**
Task spaces adaptation		
GAMP + noise	$\mathbf{0.016} \pm \mathbf{0.003}$	$\mathbf{0.025} \pm \mathbf{0.009}$
GAMP	0.019 ± 0.008	0.075 ± 0.13
Imitation	0.251 ± 0.346	1.629 ± 2.124
MLP adaptation		
GAMP + noise	0.017 ± 0.002	0.078 ± 0.014
GAMP	0.025 ± 0.008	0.086 ± 0.028
Imitation	3.2e6 $\pm$ 8.7e6	1.3e6 $\pm$ 6.4e6

created higher-dimensional dataset by randomly concatenating letters up to $\boldsymbol{\xi} \in \mathbb{R}^{32}$. Two metrics were used to check the divergence. The first one is the mean distance between the final state of the demonstrations and 10 rollouts. The second one is the standard deviation of the final state for each rollout. The rollouts were executed for twice the horizon of the demonstrations, to check if the system drifts further and by adding perturbation on the initial state. These two metrics were evaluated for 5 random concatenations of letters for each dimension. The metrics were evaluated just after initialization using the policy imitation cost and several times during 50 s of training. Results are reported in Fig. 7. They show that, as expected, high-dimensional systems tend to be more difficult to train. However, after a few seconds of optimization, no more trajectories were diverging even for high-dimensional system. They all converged within an area of at worse 0.05 m of standard deviation, while the scale of the workspace is about 1 m.

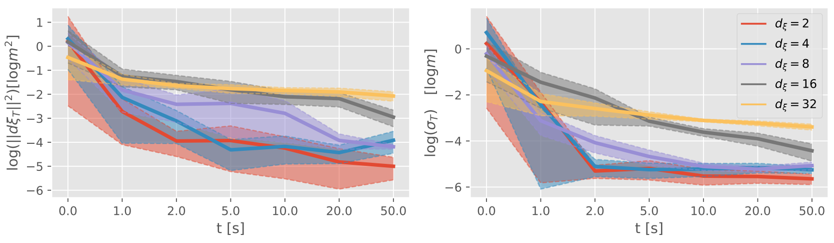


Figure 7: Evaluations of the robustness for different dimensionality of state $d_{\boldsymbol{\xi}} \in \{2, 4, 8, 16, 32\}$. (*left*) Log mean distance between the final state of the demonstrations and the 10 rollouts. (*right*) Log mean standard deviation of the final state of the 10 rollouts.