Wasserstein Gradient Flows for Optimizing Gaussian Mixture Policies

Hanna Ziesche* **and Leonel Rozo***
Bosch Center for Artificial Intelligence (BCAI)
Renningen, Germany
name.surname@de.bosch.com

Abstract

Robots often rely on a repertoire of previously-learned motion policies for performing tasks of diverse complexities. When facing unseen task conditions or when new task requirements arise, robots must adapt their motion policies accordingly. In this context, policy optimization is the *de facto* paradigm to adapt robot policies as a function of task-specific objectives. Most commonly-used motion policies carry particular structures that are often overlooked in policy optimization algorithms. We instead propose to leverage the structure of probabilistic policies by casting the policy optimization as an optimal transport problem. Specifically, we focus on robot motion policies that build on Gaussian mixture models (GMMs) and formulate the policy optimization as a Wasserstein gradient flow over the GMMs space. This naturally allows us to constrain the policy updates via the L^2-Wasserstein distance between GMMs to enhance the stability of the policy optimization process. Furthermore, we leverage the geometry of the Bures-Wasserstein manifold to optimize the Gaussian distributions of the GMM policy via Riemannian optimization. We evaluate our approach on common robotic settings: Reaching motions, collision-avoidance behaviors, and multi-goal tasks. Our results show that our method outperforms common policy optimization baselines in terms of task success rate and low-variance solutions.

1 Introduction

One of the main premises about autonomous robots is their ability to successfully perform a large range of tasks in unstructured environments. This demands robots to adapt their task models according to environment changes or new task requirements, and consequently to adjust their actions to successfully perform under unseen conditions [1]. In general, robotic tasks, such as picking or inserting an object, are usually executed by composing previously-learned skills [2], each represented by a motion policy. Therefore, in order to successfully perform under new settings, the robot should adapt its motion policies according to the new task requirements and environment conditions.

Research on methods for robot motion policy adaptation is vast [3, 4], with approaches mainly building on black-box optimizers [5, 6], end-to-end deep reinforcement learning [7, 8], and policy search [9]. Regardless of the optimization method, most approaches disregard the intrinsic policy structure in the adaptation strategy. However, several motion policy models (e.g., dynamic movement primitives (DMP)[10], Gaussian mixture models (GMM) [11], probabilistic movement primitives (ProMPs) [12, 13], and neural networks [14], among others), carry specific physical or probabilistic structures that should not be ignored. First, these policy models are often learned from demonstrations in a starting learning phase [2], thus the policy structure already encapsulates relevant prior information about the skill. Second, structure-unaware adaptation strategies optimize the policy parameters disregarding

*Equal contribution.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

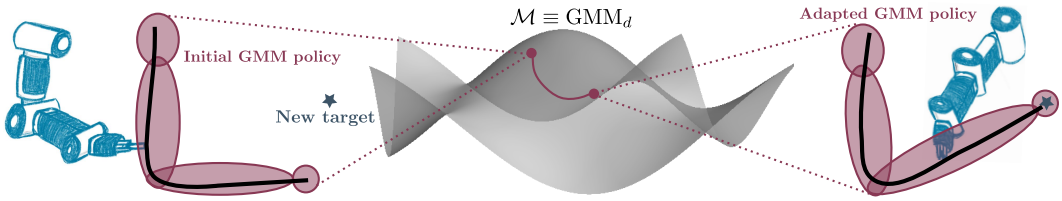


Figure 1: Illustration our policy structure-aware adaptation of GMM policies. *Left*: A robot manipulator tracks a reference trajectory (▬) extracted from a GMM policy (⬭) initially learned from demonstrations, while being required to reach for a new target (★). *Center*: Our policy optimization provides policy updates that follow a Wasserstein gradient flow on the manifold of GMM policies GMM_d. *Right*: The final GMM policy where some Gaussian components were adapted in order to retrieve a new reference trajectory that allows the robot to reach for the new target.

the intrinsic structural characteristics of the policy model (e.g., a DMP represents a second-order dynamical system). In this context, we hypothesize that the policy structure may be leveraged to better control the adaptation strategy via policy structure-aware gradients and trust regions.

Our main idea is to design a policy optimization strategy that explicitly builds on a particular policy structure. Specifically, we focus on GMM policies, which have been widely used to learn motion skills from human demonstrations [11, 15, 16, 17, 18, 19]. GMMs provide a simple but expressive enough representation for learning a large variety of robot skills such as: Stable dynamic motions [20, 21, 22], collaborative behaviors [23, 24], assembly strategies [25, 26], and contact-rich manipulation [27, 28], among others. Often, skills learned from demonstrations need to be refined — due to imperfect data — or adapted to comply with new task requirements. Existing adaptation strategies for GMM policies either build a kernel method on top of the original skill model [29], or leverage reinforcement learning (RL) to adapt the policy itself [30, 31]. However, none of these techniques explicitly considered the intrinsic probabilistic structure of the Gaussian mixture policy.

Unlike the aforementioned approaches, we propose a policy optimization technique that explicitly considers the underlying GMM structure. To do so, we exploit optimal transport theory [32, 33], which allows us to view the set of GMM policies as a particular space of probability distributions GMM_d. Specifically, we leverage the idea of Chen et al. [34] and Delon and Desolneux [35] to view a GMM as a set of discrete measures (Dirac masses) on the space of Gaussian distributions $\mathcal{G}(\mathbb{R}^d)$, which is endowed with a Wasserstein distance (see § 2). This allows us to formulate the policy optimization as a Wasserstein gradient flow (WGF) over the space of GMMs (as illustrated in Fig. 1 and explained in §3), where the policy updates are naturally guaranteed to be GMMs. Moreover, we take advantage of the geometry of the Bures-Wasserstein manifold to optimize the Gaussian distributions of a GMM policy via Riemannian optimization. We evaluate our approach over a set of different GMM policies featuring common robot skills: Reaching motions, collision-avoidance behaviors, and multi-goal tasks (see § 4). Our results show that our method outperforms common policy optimization baselines in terms of task success rate while providing low-variance solutions.

Related Work: Richemond and Maginnis [36] pioneered the idea of understanding policy optimization through the lens of optimal transport. They interpreted the policy iteration as gradient flows by leveraging the implicit Euler scheme under a Wasserstein distance (see § 2), considering only 1-step return settings. They observed that the resulting policy optimization resembles the gradient flow of the Fokker-Planck equation (JKO scheme) [37]. In a similar spirit, Zhang et al. [38] proposed to use WGFs to formulate policy optimization as a sequence of policy updates traveling along a gradient flow on the space of probability distributions until convergence. To solve the WGF problem, the authors proposed a particle-based algorithm to approximate continuous density functions and subsequently derived the gradients for particle updates based on the JKO scheme. Although Zhang et al. [38] considered general parametric policies, their method assumed a distribution over the policy parameters and did not consider a specific policy structure, which partially motivated their particle-based approximation. Recently, Mokrov et al. [39] tackled the computational burden of particle methods by leveraging input-convex neural networks to approximate the WGFs computation. They reformulated the well-known JKO optimization [37] over probability measures by an optimization over convex functions. Yet, this work remains a general solution for WGF computation and it did not address its use for policy optimization problems.

Aside from optimal transport approaches, Arenz et al. [30] proposed a trust-region variational inference for GMMs to approximate multimodal distributions. Although not originally designed

for policy optimization, the authors elucidated a connection to learn GMMs of policy parameters in black-box RL. However, their method cannot directly be applied to our GMM policy adaptation setting, nor does it consider the GMM structure from an optimal transport perspective. Nematollahi et al. [31] proposed SAC-GMM, a hybrid model that employs the well-known SAC algorithm [40] to refine dynamic skills encoded by GMMs. The SAC policy was designed to learn residuals on a single vectorized stack of GMM parameters, thus fully disregarding the GMM structure and the geometric constraints of its parameters. Ren et al. [41] introduced PMOE, a method to train deep RL policies using a probabilistic mixture of experts via GMMs. They addressed the non-differentiability problem caused by the optimization of categorical distribution parameters associated with the mixture weights via gradient estimators. However, PMOE does not take an optimal transport perspective on the policy optimization problem, nor does it account for the geometry arising from the GMM parameters.

Concerning approaches that explicitly account for the geometry induced by the policy, we may broadly categorize them into two groups: exact models and natural gradient methods. The former category refers to policy optimization methods that fully exploit the geometric structure of the policy, as proposed in this paper. In this case, the state of the art is scarce but recent works such as the Riemannian proximal policy optimization for GMMs proposed in [42], showed the importance of leveraging the geometry induced by the GMM parameters in the policy optimization via Riemannian gradients, similarly to our method. Their policy optimization was regularized by a Wasserstein distance to control the exploration-exploitation trade-off. However, their method did not formulate the policy optimization as an optimal transport problem, i.e., the policy updates did not follow a Wasserstein gradient flow, as in our approach, but it employed instead a classical non-convex optimization procedure.

The second category encloses approaches that leverage the well-established technique for enhancing the stability of policy optimization, namely, introducing regularization to the objective function using the Kullback-Leibler (KL) divergence, which induces the so-called natural gradient [43, 44, 45, 46]. In this context, the regularization provides a global measure of the policies similarity, while the natural gradient perspective aims at locally controlling the changes between successive policy updates. In other words, natural gradient approaches leverage the local geometry of the policy parameters space to compute controlled updates. Note that policies similarity can also be quantified via the Wasserstein distance, as proposed in this paper and in recent works [36, 38, 47, 48]. Unlike the KL divergence, the Wasserstein distance enjoys powerful geometrical, computational, and statistical features [32, 33]. For example, Moskovitz et al. [48] recently employed the Wasserstein natural gradient to exploit the local geometry induced by the Wasserstein regularization of behavioral policy optimization [47], but neither the policy nor the behavioral embeddings had particular structures that could be leveraged in the policy optimization.

In this paper we exploit the geometric properties arising from both a Wasserstein-regularized objective and a Gaussian mixture policy. This allows us not only to see the policy iteration as a Wasserstein gradient flow, as in [36], but also to leverage the Riemannian geometry associated to the GMM space GMM_d to formulate exact Riemannian gradient descent updates [49], instead of relying on inexact natural gradients approximations. To achieve this, our method leverages the geometry induced by the structure of the space of GMM policies via the Bures-Wasserstein manifold, which naturally guarantees that policy updates stay on GMM_d. To the best of our knowledge, our optimization of GMM policies based on Wasserstein gradient flows and the Bures-Wasserstein manifold is the first of its kind in the domain of policy adaptation.

2 Background

2.1 Wasserstein gradient flows

In Euclidean space a gradient flow is a smooth curve $x : \mathbb{R} \to \mathbb{R}^d$ that satisfies the partial differential equation (PDE) $\dot{x}(t) = -\nabla L(x(t))$ for a given loss function $L : \mathbb{R}^d \to \mathbb{R}$ and starting point x_0 at $t = 0$ [32, 50]. A solution can be found straightforwardly by forward discretization, leading to the well-known explicit Euler update scheme $x^\tau_{k+1} = x^\tau_k - \lambda \nabla L(x^\tau_k)$, where λ denotes the learning rate and x^τ indicates a discretization of the curve $x(t)$ with discretization parameter τ. Alternatively, we can use a backward discretization, which leads to the following implicit Euler scheme

$$x^\tau_{k+1} = \arg\min_x \left(\frac{\|x - x^\tau_k\|^2}{2\tau} + L(x) \right). \tag{1}$$

Eq. 1 is sometimes referred to as Minimizing Movement Scheme and can be used as an alternative characterization of a gradient flow.

This characterization is particularly interesting when we need to extend the concept of gradient flows to (non-Euclidean) general metric settings, since there is no notion of ∇L in these cases [32, 51]. Eq. 1 does not involve any gradients and can be expressed using only metric quantities. In this work, we are particularly interested in gradient flows in the L^2-Wasserstein space, defined as the set of probability measures $\mathbb{P}(\mathcal{X})$ on a separable Banach space $\mathcal{X}$ [52] and endowed with the L^2-Wasserstein distance W_2 defined as

$$W_2(\mu, \nu) = \left(\inf_{\gamma \in \Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{X}} \|x_1 - x_2\|^2 \mathrm{d}\gamma(x_1, x_2) \right)^{\frac{1}{2}}, \tag{2}$$

where $\mu, \nu \in \mathbb{P}(\mathcal{X})$ and $\gamma \in \mathbb{P}(\mathcal{X}^2)$ is defined to have the two marginals μ and ν.

A Generalized Minimizing Movement scheme characterizing gradient flows in the Wasserstein space can be written in analogy to Eq. 1 as:

$$\pi^\tau_{k+1} = \arg\min_\pi \left(\frac{W_2^2(\pi, \pi^\tau_k)}{2\tau} + L(\pi) \right), \tag{3}$$

where L is a functional to be minimized on the Wasserstein space and $\pi_k \in \mathbb{P}(X)$. In the following, we will omit the superscript τ for notational convenience.

2.2 Reinforcement Learning as Wasserstein Gradient Flows

Our view of the policy structure-aware optimization builds on the approach outlined in [36], which in turn is based on the JKO scheme [37]. They proposed a formulation of 1-step RL problems in terms of Wasserstein gradient flows. In particular, they studied the evolution of a policy π under the influence of a free energy functional J of the form:

$$J(\pi) = K_r(\pi) + \beta\mathcal{H}(\pi) = \int_{\mathcal{A}} \mathrm{d}\pi(\boldsymbol{a}|\boldsymbol{s}) r(\boldsymbol{s}, \boldsymbol{a}) - \beta \int_{\mathcal{A}} \mathrm{d}\pi(\boldsymbol{a}|\boldsymbol{s}) \log(\pi(\boldsymbol{a}|\boldsymbol{s})), \tag{4}$$

where $K_r(\pi)$ denotes the inner energy of the system, here determined by the reward $r(\boldsymbol{s}, \boldsymbol{a})$, and $\mathcal{H}(\pi)$ is the entropy of the policy $\pi(\boldsymbol{a}|\boldsymbol{s})$, with $\boldsymbol{s}$ and $\boldsymbol{a}$ denoting the state and action, respectively. Thus, Eq. 4 can be recognized as the usual objective in 1-step RL settings with entropy regularization. It is well known that the evolution of probability densities under a free energy of this form is properly described by a PDE known as the Fokker-Planck equation. Richemond and Maginnis [36] exploited the result of Jordan et al. [37], which stated that this evolution can be interpreted as the gradient flow of the functional J in Wasserstein space. This flow is characterized by the following minimizing movement scheme

$$\pi_{k+1} = \arg\min_\pi \left(\frac{W_2^2(\pi, \pi_k)}{2\tau} - J(\pi) \right), \tag{5}$$

which naturally provides iterative updates for the policy π. While Richemond and Maginnis [36] considered a 1-step bandit setting, we extend this approach to full multi-step RL problems and consequently learn policies for long-horizon tasks.

2.3 The L^2-Wasserstein distance between Gaussian Mixture Models (GMMs)

We consider policies $\pi(\boldsymbol{x})$ that build on a GMM structure, i.e., $\pi(\boldsymbol{x}) = \sum_{i=1}^N \omega_i \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, where $\mathcal{N}$ denotes a multivariate Gaussian distribution with mean $\boldsymbol{\mu}_i$ and covariance matrix $\boldsymbol{\Sigma}_i$, and ω_i are the weights of the N individual Gaussian components, which are subject to $\sum_i \omega_i = 1$. In the following, we will write $\hat{\boldsymbol{\mu}}$, $\hat{\boldsymbol{\Sigma}}$ and $\hat{\boldsymbol{\omega}}$ to denote the stacked means, covariance matrices and weights of the N components. Therefore, we do not consider WGFs on the full manifold of probability distributions (Wasserstein space) $\mathbb{P}(\mathbb{R}^d)$ but rather focus on WGFs evolving on the submanifold of GMMs, that is $\mathrm{GMM}_d \subset \mathbb{P}(\mathbb{R}^d)$. Following [34, 35], we can approximately describe this submanifold as a discrete distribution over the space of Gaussian distributions equipped with the Wasserstein metric. This in turn can be identified with the Bures-Wasserstein manifold which is the product manifold $\mathbb{R}^d \times \mathcal{S}^d_{++}$,

where $\mathcal{S}^d_{++}$ denotes the Riemannian manifold of d-dimensional symmetric positive definite matrices. The corresponding approximated Wasserstein distance between two GMMs π_1, π_2 is given by

$$W_2^2\big(\pi_1(\boldsymbol{x}), \pi_2(\boldsymbol{x})\big) = \min_{\boldsymbol{P} \in U(\boldsymbol{\omega}_1, \boldsymbol{\omega}_2)} \sum_{i,j}^{N} \boldsymbol{P}_{ij} W_2^2\big(\mathcal{N}_1(\boldsymbol{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), \mathcal{N}_2(\boldsymbol{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\big), \tag{6}$$

where $U(\boldsymbol{\omega}_1, \boldsymbol{\omega}_2) = \{\boldsymbol{P} \in \mathbb{R}_+^{N \times N} | \boldsymbol{P}\mathbf{1}_N = \boldsymbol{\omega}_1, \boldsymbol{P}^\mathsf{T}\mathbf{1}_N = \boldsymbol{\omega}_2\}$ with $\mathbf{1}_N$ denoting an N-dimensional vector of ones. The Wasserstein distance between two Gaussian distributions in Eq. 6 can be computed analytically as follows

$$W_2^2\big(\mathcal{N}_1(\boldsymbol{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), \mathcal{N}_2(\boldsymbol{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\big) = \|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|^2 + \operatorname{tr}\left[\boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_j - 2\left(\boldsymbol{\Sigma}_i^{1/2}\boldsymbol{\Sigma}_j\boldsymbol{\Sigma}_i^{1/2}\right)^{1/2}\right]. \tag{7}$$

2.4 Learning GMM policies from demonstrations

A popular approach in RL — particularly in robotics — to reduce the number of policy rollouts in the environment is to warm-start the policy with a set of demonstrations provided by an expert. In this work we choose to represent our policy via a GMM. We assume that demonstrations are provided as a set of trajectories τ of state-action pairs $\tau = \{(\boldsymbol{s}_0, \boldsymbol{a}_0), (\boldsymbol{s}_1, \boldsymbol{a}_1), \dots (\boldsymbol{s}_T, \boldsymbol{a}_T)\}$. To initialize our policy, we first use the Expectation-Maximization (EM) algorithm to fit a GMM, in the state-action space, to the demonstrations. This results in a mixture distribution $\pi(\boldsymbol{s}, \boldsymbol{a}) = \sum_{i=1}^{N} \omega_i \mathcal{N}\big([\boldsymbol{s}\, \boldsymbol{a}]^\mathsf{T}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\big)$ from which a policy can be obtained by conditioning on the state $\boldsymbol{s}$, as follows

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \frac{\pi(\boldsymbol{s}, \boldsymbol{a})}{\int \pi(\boldsymbol{s}, \boldsymbol{a}) \mathrm{d}\boldsymbol{a}}. \tag{8}$$

In the context of GMMs, this is also known as Gaussian Mixture Regression (GMR) [53]. The resulting conditional distribution is another GMM on action space with state-dependent parameters,

$$\pi(\boldsymbol{a}_t|\boldsymbol{s}_t) = \sum_{i=1}^{N} \omega_i(\boldsymbol{s}_t) \mathcal{N}(\boldsymbol{a}_t; \boldsymbol{\mu}_i^a(\boldsymbol{s}_t), \boldsymbol{\Sigma}_i^a). \tag{9}$$

Details on computation of Eq. 9 from the original GMM are given in App. A.1.

3 Wasserstein Gradient Flows for GMM Policy Optimization

In this work, we focus on multi-step RL tasks for policy adaptation. We consider a finite-horizon Markov Decision Process (MDP) with continuous state and action spaces $\mathcal{S} \in \mathbb{R}^n$ and $\mathcal{A} \in \mathbb{R}^m$, transition and reward functions $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ and $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$, initial state distribution $\rho(\boldsymbol{s}_0)$ and a discount factor γ. Further, we assume to have an initial policy $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$, which is to be adapted by optimizing some objective function $K_r(\pi)$. As stated in § 1, this problem arises in robot learning settings where a policy learned via imitation learning (e.g., LfD) needs to be adapted to new objectives or unseen environmental conditions. To promote exploration and avoid premature convergence to suboptimal policies, we leverage maximum entropy RL [54] by adding an entropy term $\mathcal{H}(\pi)$ to the objective. Thus, the overall objective has the form of a free energy functional (resembling Eq. 4) and can be written as

$$J(\pi) = K_r(\pi) + \beta\mathcal{H}(\pi), \tag{10}$$

where β is a hyperparameter and $K_r(\pi)$ corresponds to the usual cumulative return

$$K_r(\pi) = \mathbb{E}_\tau\left[\sum_t r(\boldsymbol{s}_t, \boldsymbol{a}_t)\right] = \int \Pi_t \mathrm{d}\boldsymbol{s}_0 \mathrm{d}\boldsymbol{s}_t \mathrm{d}\boldsymbol{a}_t \rho(\boldsymbol{s}_0)\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)\sum_t \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t). \tag{11}$$

The evolution of the policy $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ over the course of the optimization can be described as a flow of a probability distribution in Wasserstein space. This formulation comes with three major benefits: *(i)* We directly leverage the Wasserstein metric properties for describing the evolution of probability distributions; *(ii)* We exploit the L^2-Wasserstein distance to constrain the policy updates, which is important to guarantee stability in policy optimization [55, 45, 56]; *(iii)* By constraining to specific

submanifolds of the Wasserstein space, in this case GMMs, we can impose additional structural properties on the policy optimization.

Since our objective in Eq. 10 has the form of the free energy functional studied by [36, 37], we can leverage the iterative updates scheme of Eq. 5 to formulate the evolution of our policy iteration under the flow generated by Eq. 10. As mentioned previously, we focus on the special case of GMM policies and therefore restrict the Wasserstein gradient flow to the submanifold of GMM distributions GMM_d. We refer the interested reader to App. A.3, where we provide the explicit form of $J(\pi)$ of Eq. 10 for the GMM case.

3.1 Policy optimization

To begin with, we leverage the approximation that describes the GMM submanifold as a discrete distribution over the space of Gaussian distributions $\mathcal{G}(\mathbb{R}^d)$, equipped with the Wasserstein metric [34, 35]. Consequently, our policy optimization problem naturally splits into an optimization over the $(N-1)$-dimensional simplex and an optimization on the N-fold product of the Bures-Wasserstein manifold (BW^N), i.e., the product manifold $\left(\mathbb{R}^d \times \mathcal{S}^d_{++}\right)^N$. The former corresponds to the GMM weights while the latter applies to the set of Gaussian distributions' parameters. Note that the identification with the BW^N manifold allows us to perform the optimization directly on the parameter space. This comes with several benefits: *(i)* We can leverage the well-known analytic solution of the Wasserstein distance between Gaussian distributions in Eq. 6, greatly reducing the computational complexity of the policy optimization. *(ii)* As shown in [34], we can guarantee that the policy optimized via Eq. 6 remains a GMM (i.e., it satisfies the mass conservation constraint). *(iii)* Unlike the full Wasserstein space[2], the resulting product manifold is a true Riemannian manifold such that we can leverage the machinery of Riemannian optimization. Importantly, working in the parameter space allows us to apply an explicit Euler scheme, instead of the implicit formulation of Eq. 3, when optimizing the Gaussian distributions' parameters.

According to the above splitting scheme, we formulate the policy optimization as a two-step procedure that alternates between the Gaussian parameters (i.e. means and covariance matrices) and the GMM weights. To optimize the former, we propose to leverage the Riemannian structure of the BW manifold to reformulate the updates as a forward discretization, similarly to [58]. In other words, by embedding the Gaussian components of the GMM policy in a Riemannian manifold, the Wasserstein gradient flow in the implicit form of Eq. 5 can be approximated by an explicit Euler update scheme according to the BW metric (further details are provided in App. A.4). This allows us to leverage the expressions of the Riemannian gradient and exponential map of the BW manifold [59, 60]. Thus, the optimization boils down to Riemannian gradient descent where the gradient is defined w.r.t the Bures-Wasserstein metric. In particular, we use the expression for Riemannian gradient, metric and exponential map used in [60]. Formally, the resulting updates for the Gaussian parameters of the GMM follow the Riemannian gradient descent scheme given by:

$$\hat{\boldsymbol{\mu}}_{k+1} = \mathrm{R}_{\hat{\boldsymbol{\mu}}_k}\left(\lambda \cdot \mathrm{grad}_{\hat{\boldsymbol{\mu}}} J(\pi_k)\right), \quad \text{and} \quad \hat{\boldsymbol{\Sigma}}_{k+1} = \mathrm{R}_{\hat{\boldsymbol{\Sigma}}_k}\left(\lambda \cdot \mathrm{grad}_{\hat{\boldsymbol{\Sigma}}} J(\pi_k)\right), \tag{12}$$

where grad denotes the Riemannian gradient w.r.t. the Bures-Wasserstein metric, $\mathrm{R}_{\boldsymbol{x}} : \mathcal{T}_{\boldsymbol{x}}\mathcal{M} \to \mathcal{M}$ denotes the retraction operator, which maps a point on the tangent space $\mathcal{T}_{\boldsymbol{x}}\mathcal{M}$ back to the manifold $\mathcal{M} \equiv \mathrm{BW}$ [61]. Moreover, λ is a learning rate and $\pi_k \stackrel{\text{def}}{=} \pi(\hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k, \hat{\boldsymbol{\omega}}_k)$. The Euclidean gradients of $J(\pi)$ required for computing grad can be obtained using a likelihood ratio estimator (a.k.a score function estimator or REINFORCE) [62] and are provided in App. A.3.

Concerning the GMM weights, we first reparameterize them as $\omega_j = \frac{\exp \eta_j}{\sum_{k=1}^N \exp \eta_k}$ and optimize w.r.t. the new parameters $\eta_j, j = 1...N$, which unlike $\hat{\boldsymbol{\omega}}$ are unconstrained. For this optimization we employ the implicit Euler scheme:

$$\hat{\boldsymbol{\eta}}_{k+1} = \arg\min_{\hat{\boldsymbol{\eta}}} \left(\frac{W_2^2(\pi_{k+1}(\hat{\boldsymbol{\eta}}), \pi_k)}{2\tau} - J(\pi_{k+1}(\hat{\boldsymbol{\eta}})) \right), \tag{13}$$

[2]Wasserstein space is not a true Riemannian manifold, but it can be equipped with a Riemannian structure and formal calculus on this manifold [57], which has been made rigorous in [51]

Algorithm 1 GMM Policy Optimization via Wasserstein Gradient Flows

Input: initial policy $\pi(\boldsymbol{a}|\boldsymbol{s})$

while not goal reached **do**
  Rollout policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ in the environment for M episodes to collect interaction trajectories $\tau = \{(\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{r}_0), (\boldsymbol{s}_1, \boldsymbol{a}_1, \boldsymbol{r}_1), \ldots, (\boldsymbol{s}_T, \boldsymbol{a}_T, \boldsymbol{r}_T)\}_{m=1}^{M}$
  repeat
    Update Gaussian components parameters $\hat{\boldsymbol{\mu}}$, $\hat{\boldsymbol{\Sigma}}$ using Riemannian optimization (12), where λ^{ls} is determined via line-search (see §3.2).
  until convergence
  repeat
    Update GMM weights $\hat{\boldsymbol{\omega}}$ via gradient descent on the free energy objective 10, using 14
  until converged
end while

where $\pi_{k+1}(\hat{\boldsymbol{\eta}}) \stackrel{\text{def}}{=} \pi(\hat{\boldsymbol{\mu}}_{k+1}, \hat{\boldsymbol{\Sigma}}_{k+1}, \hat{\boldsymbol{\eta}})$. We minimize Eq. 13 by gradient descent w.r.t. $\boldsymbol{\eta}$ as follows:

$$\hat{\boldsymbol{\eta}}_{k+1} = \hat{\boldsymbol{\eta}}_k - \lambda \nabla_{\hat{\boldsymbol{\eta}}} \left(\frac{W_2^2(\pi_{k+1}(\boldsymbol{\eta}), \pi_k)}{\tau} - J(\pi_{k+1}(\hat{\boldsymbol{\eta}})) \right). \tag{14}$$

The gradient of $J(\pi)$ can be obtained analytically using a likelihood ratio estimator. For the Wasserstein term, we first compute the gradient w.r.t. the weights via the Sinkhorn algorithm [63], from which the gradient w.r.t $\boldsymbol{\eta}$ can be then obtained via the chain rule. Note that we have to rely on the Sinkhorn algorithm here since there is no analytic solution available for the Wasserstein distance between discrete distributions, unlike the above case of the Gaussian components. Consequently, we cannot compute the corresponding gradients.

3.2 Implementation Pipeline

To carry out the policy optimization, we proceed as in the usual on-policy RL scheme: We first roll out the current policy to collect samples of state-action-reward tuples. Then, we use the collected interaction trajectories to compute a sample-based estimate of the functional $K_r(\pi)$ and its gradients w.r.t the policy parameters, as explained in § 3.1. An optimization step consists of alternating between optimizing the Gaussian parameters using Eq. 12, and updating the weights via Eq. 14. For the optimization of the Gaussian parameters we leverage Pymanopt [64] for Riemannian optimization. We extended this library by implementing the Bures-Wasserstein manifold based on the expressions provided by Han et al. [60] (see App. A.2 for details). Furthermore, we added a custom line-search routine that accounts for the constraint on the Wasserstein distance between the old and the optimized GMM, as to our knowledge such a search method does not exist in out-of-the-box optimizers. The details of this custom line-search are given in Algorithm 2 in App. A.5. Regarding the optimization of the GMM weights, we use POT [65], a Python library for optimal transport, from which we obtain the quantities required to compute the gradients of the Wasserstein distance w.r.t. the weights in Eq. 14.

The full policy optimization finishes if either the objective stops improving or the Wasserstein distance between the old and optimized GMMs exceeds a predefined threshold, which is chosen experimentally. Afterwards, fresh rollouts are performed with the updated policy and the aforementioned two-step alternating procedure starts over. This optimization loop is repeated until a task-specific success criterion has been fulfilled. We summarize the proposed optimization in Algorithm 1.

4 Experiments

We tested our approach in three different robotic settings: a reaching skill, a collision-free trajectory tracking, and a multiple-goal task. All these tasks were represented in a 2D operational space. Moreover, we tested our approach on two additional versions of the collision-free trajectory tracking task that are formulated for 3D operational-space and 7D joint-space representations of the end-effector state and action. These two latter tasks were performed by a 7-DoF Franka Emika Panda robot in a virtual environment as reported in App. A.6.3. All robot motion policies were initially learned from human demonstrations collected on a Python GUI. We assumed we were given M demonstrations, each of which contained T_m data points for a dataset of $T = \sum_m T_m$ total observations

$\tau = \{(\boldsymbol{s}_t, \boldsymbol{a}_t)\}_{t=1}^{T}$. The state $\boldsymbol{s}$ and action $\boldsymbol{a}$ corresponded to the robot end-effector position $\boldsymbol{x} \in \mathbb{R}^d$ and velocity $\dot{\boldsymbol{x}} \in \mathbb{R}^d$, with d being the operational space dimensionality. The GMM models were initially trained via classical Expectation-Maximization. The policy rollout consisted of sampling a velocity action $\boldsymbol{a}_t \sim \pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ using Eq. 9, and subsequently commanding the robot via a Cartesian velocity controller at a frequency of 100Hz. For all the experiments, we used the Robotics Toolbox for Python [66] to simulate the robotic environments.

To show the importance of accounting for the policy structure in RL settings, we compared our method against two structure-unaware baselines: PPO [45] and SAC-GMM [31]. Our baselines choice is motivated by the fact that these two RL methods are still widely used and fairly competitive [67], even in real robotic scenarios [68]. We also considered an additional baseline: PMOE [41], which is a method to train deep RL policies using a probabilistic mixture of experts via GMMs. As PPO was not originally designed to directly optimize the parameters of a previously-learned GMM policy, we designed the policy actions to represent (usually small) corrections to the GMM parameters, i.e. $\boldsymbol{a} = [\Delta\boldsymbol{\omega}\ \Delta\hat{\boldsymbol{\mu}}\ \Delta\,\text{vec}(\hat{\boldsymbol{\Sigma}})]$, following the same methodology as SAC-GMM [31]. The PPO and SAC implementations correspond to the code provided by Stable-Baselines3 [69], whose policies are parametrized by MLP networks. The PMOE implementation corresponds to the code given in the original paper [41]. During policy optimization, we sampled an action from the MLP policy that was then used to update the GMM parameters by adding the computed corrections to the current parameters. Later, we proceeded as described earlier, namely, the updated GMM was used to compute the velocity action via Eq. 9. For comparison purposes, we report statistical results for all the settings over 5 runs for the task success rate and solution variance. We tuned the baselines separately for each task using Optuna [70]. In addition, to assess the importance of our Riemannian formulation, we performed an ablation where we used the implicit scheme based on Euclidean gradient descent instead of the explicit optimization on the Bures-Wasserstein manifold (see App. A.6.2).

4.1 Tasks description

Reaching Task: This experiment consists of: (1) learning an initial GMM policy such that the robot end-effector reaches a target by following an L-shape trajectory from its initial position, and (2) adapting the GMM policy to reach a new target located midway and above the previously-learned L-shape trajectories. The initial policy, shown in Fig. 2-*left* and Fig. 13a, was learned from 12 demonstrations and encoded by a 7-component GMM. To adapt the policy, we defined a dense reward as a function of the position error between the robot end-effector and the new target. We also added a sparse penalty term that punishes rollouts leading to significantly divergent trajectories. Each optimization episode comprised 10 rollouts, each of maximum horizon length of 200 iterations. Convergence is achieved when a minimum average position error w.r.t the target – computed over an episode – is reached.

Collision-avoidance Task: This task consists of: (1) learning an initial GMM policy of a linear reaching motion, and (2) adapting the GMM policy to reach a new horizontally-translated target while avoiding to collide with two spherical obstacles located midway between the initial robot position and the new target. The initial GMM policy was learned from 10 human demonstrations and represented by a 3-component GMM, as shown in Fig. 2-*middle* and Fig. 13b. For policy optimization, we defined a sparse reward as a function of the position error between the robot end-effector position and the target at the end of the rollout. We also included two sparse penalty terms: the first one punishes rollouts leading to collisions with the obstacles, for which the rollout is stopped; the second term penalizes rollouts with significantly divergent trajectories. Each episode consisted of 10 rollouts, each of maximum horizon length of 150 iterations. Convergence is determined by a minimum average position error w.r.t the target computed over an episode.

Multiple-goal Task: This setting involves: (1) learning an initial GMM policy where the robot end-effector reaches two different targets (i.e., task goals) starting from the same initial position, and (2) adapting the initial policy to reach a new target located close to one of the previous task goals. The intended adapted behavior should make the robot go through the most relevant GMM components according to the new target location. The initial GMM policy was learned from 12 demonstrations and encoded by a 6-component GMM, as shown in Fig. 2-*right* and Fig. 13c. To optimize the initial GMM policy, we specified a sparse reward based on the position error between the robot end-effector position and the chosen target at the end of the rollout. Similar to the previous experiments, we added a sparse penalty term to penalize rollouts generating significantly divergent trajectories. An

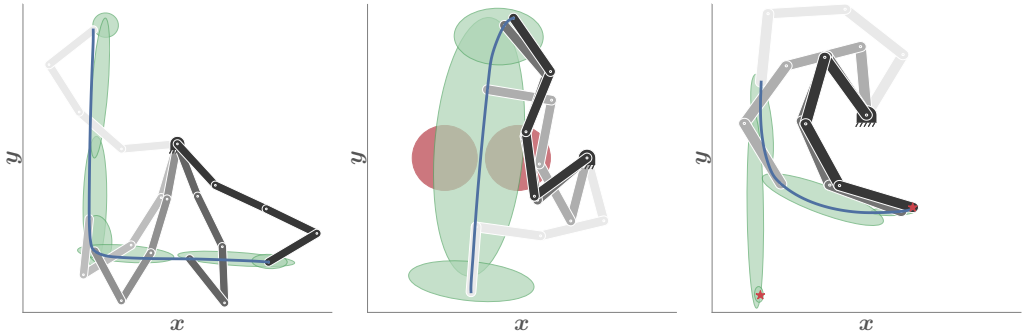


Figure 2: The three tested 2D robotic settings: a reaching skill (*left*), a collision-free trajectory tracking (*middle*), and a multiple-goal task (*right*). The robot color goes from light gray to black to show the evolution of the task reproduction. Green Gaussian components (⬭) depict the initial GMM policy, projected on the 2D Cartesian position space. The end-effector trajectory resulting from the initial GMM policy is shown in dark blue lines (—). Red circles (●) in the collision-avoidance task represent the obstacles (*middle*). The different targets of the multiple-goal task (*right*) are depicted as red stars.

episode comprised 10 rollouts, each of maximum horizon length of 200 iterations. Again, the policy optimization converges when the average position error w.r.t the chosen target reaches a minimum threshold.

4.2 Results Analysis

The reaching task tested our method's ability to adapt a previously-learned reaching skill to a new goal, located at $(6.0, -6.5)$ (cf. Fig. 13a-*left*). Achieving this required to adapt the Gaussian parameters of mainly the last four GMM components, while the others remained unchanged. We compared all methods in terms of the success rate over environment steps, where the success rate was defined as the percentage of rollouts reaching the new goal. Figure 3-*left* shows that our method achieved a success rate of 1 after approximately 70000 environment interactions. Despite PPO was also able to complete the task reliably, it required many more environment steps (cf. Fig. 5-*left*). In sharp contrast, SAC did not reach any improvement. These observations underline the importance of some kind of trust region or constraint on the policy updates, which allowed both our method and PPO to reach good success rates. The PMOE versions of PPO and SAC showed a better performance than the vanilla counterparts, which aligns with the experimental insights provided in the original PMOE paper [41] regarding the advantages of the probabilistic mixture of experts in deep RL policies. Nevertheless, our approach also outperformed both PMOE versions. This experiment showed that our method is much more sample-efficient in adapting the GMM parameters, which we attribute to the fact that our method explicitly takes the GMM structure into account in the formulation of the optimization.

In the collision-avoidance task, we tested whether our method was able to adapt a trajectory tracking skill in order to avoid collisions with newly added obstacles. These were placed in such a way that the robot was forced to move its end-effector through a narrow path between the obstacles (cf. Fig. 2-*middle*). While the reaching task could be adapted by mainly varying the means of the GMM components, this task also demands to adapt the covariance of the second GMM component. Figure 3-*middle* shows that our method solved this task reliably after comparatively few environment interactions. Although PPO also achieved a success rate of 1, it took 6 times more environment steps than our method. SAC only reached an average success rate of 0.8, however with high variance (cf. Fig. 5-*middle*). Note that the PMOE versions showed a reduced solution variance and a higher success rate than their vanilla counterparts, where PMOE-PPO also outperformed PMOE-SAC. These results again show the importance of the constraints on the policy updates. The huge discrepancy in the required environment steps between our method and all the baselines further emphasizes the importance of taking the GMM structure into account in the policy optimization. The explicit consideration of this structure allows our method to identify the required changes to the GMM parameters much more efficiently. These experimental insights can also be observed in the results corresponding to the other formulations of the collision-avoidance task, namely, the 3D operational and 7D joint space representations, reported in App. A.6.3.

While the previous two tasks were accomplished by adapting mostly the Gaussian parameters of the GMM, the multiple-goal task required to adapt the GMM weights. The initial skill comprised

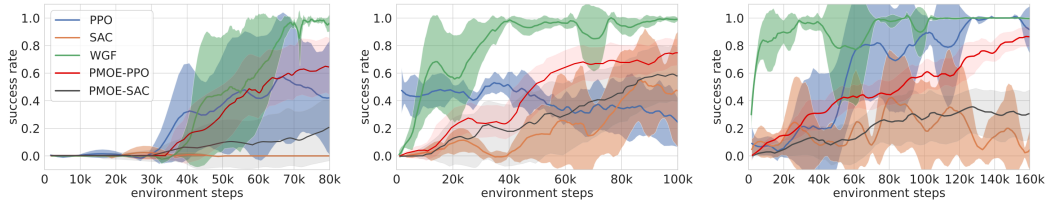


Figure 3: Success rate of our method (WGF) and the baselines on the reaching (*left*), the collision-avoidance (*middle*) and the multiple-goal tasks (*right*). The shaded area depicts the standard deviation over 5 runs.

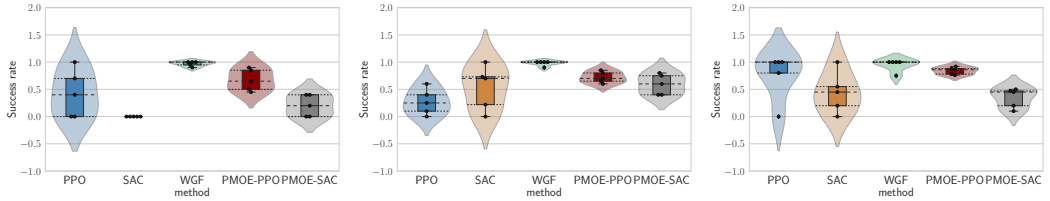


Figure 4: Variance of the success rate over the 5 runs for our method (WGF) and the baselines on the reaching task (*left*), the collision avoidance task (*middle*) and the multiple-goal task (*right*). The violine plots are overlaid with box plots, quartile lines and a swarm plot, where dots indicate the success rates of individual runs. The plots show the variance at the following time steps from left to right: 80000, 90000, 95000.

reaching motions to two different goals and an execution of this skill results in reaching one of them, depending on the sampling noise (cf. Fig. 13c). The easiest way to adapt the policy to reach only one of the two goals is to reduce the GMM weights of the components belonging to the undesired motion and correspondingly increase the weights of the other components. As shown in Fig. 3-*right*, our method again quickly achieved a success rate of 1. PPO required substantially many more environment steps, while SAC was not able to solve the task. Again, the PMOE formulation reduced the solution variance and slightly increased the success rate of both PPO and SAC.

In Fig. 4 we report the success rate variance over 5 runs at a fixed time step, which corresponded to the step at which the first method achieved a success rate of 1, thus prioritizing sample efficiency. The plots show that our method exhibits a very low solution variance. All baselines varied largely, except for the reaching task, where all SAC runs collapsed to a success rate of 0. These results show that our method, despite showing large variance at the start, was able to quickly reduce the variance and converge reliably to a good success rate. We also provide similar plots of solution variance in Fig. 6, where we report the results for both PPO and SAC using their own convergence time step.

5 Conclusions and Future Work

We presented a novel method for GMM policy optimization, which leverages optimal transport theory to formulate the policy optimization as a Wasserstein gradient flow on the manifold of GMMs. Our formulation explicitly accounts for the GMM structure in the optimization and furthermore enables us to naturally constrain the policy updates by the L^2-Wasserstein distance between GMMs to enhance the stability of the policy optimization process. Moreover, the embedding of the Gaussian components of the GMM policy in the Bures-Wasserstein manifold greatly reduced the computational cost of the policy optimization. Experiments on several robotic tasks provided strong evidence of the importance of our policy-structure aware optimization against approaches that disregard the GMM structure. A possible limitation of our method is that each optimization loop involves running the Sinkhorn algorithm, which is computationally expensive. This might be improved by employing recent advances on initializing the Sinkhorn algorithm [71]. Also, we observed an intricate interplay between the optimization of the GMM weights and the Gaussian parameters, which occasionally resulted in one update hampering the other. In future work we plan to address the latter problem by using separate adaptive learning rates for weights and Gaussian parameters. Another possibility would entail to reformulate the policy optimization as a gradient flow on the space of probability measures endowed with the Wasserstein Fisher-Rao metric [72, 73, 74]. This alternative metric may allow us to leverage the Fisher-Rao geometry to optimize the GMM weights, as very recently proposed in [75] to learn isotropic Gaussian mixtures via particle-based approximations. Finally it would be interesting to combine our method with an actor-critic formulation and to replace the multi-step cumulative reward by a trained Q-function.

References

[1] Jan Peters, Daniel D. Lee, Jens Kober, Duy Nguyen-Tuong, J. Andrew Bagnell, and Stefan Schaal. *Robot Learning*, pages 357–398. Springer International Publishing, 2016. ISBN 978-3-319-32552-1. URL https://doi.org/10.1007/978-3-319-32552-1_15. 1

[2] Stefan Schaal, Auke Ijspeert, and Aude Billard. Computational approaches to motor learning by imitation. *Phil. Trans. R. Soc. Lond. B*, 358:537–547, 2003. URL http://doi.org/10.1098/rstb.2002.1258. 1

[3] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research (IJRR)*, 32(11):1238–1274, 2013. URL https://doi.org/10.1177/0278364913495721. 1

[4] Konstantinos Chatzilygeroudis, Vassilis Vassiliades, Freek Stulp, Sylvain Calinon, and Jean-Baptiste Mouret. A survey on policy search algorithms for learning robot controllers in a handful of trials. *IEEE Transactions on Robotics*, 36(2):328–347, 2020. URL https://doi.org/10.1109/TRO.2019.2958211. 1

[5] Freek Stulp and Olivier Sigaud. Policy improvement methods: Between black-box optimization and episodic reinforcement learning. hal-00738463, 2012. URL http://hal.archives-ouvertes.fr/hal-00738463. 1

[6] Noémie Jaquier, Leonel Rozo, Sylvain Calinon, and Mathias Bürger. Bayesian optimization meets Riemannian manifolds in robot learning. In *Conference on Robot Learning (CoRL)*, pages 233–246, 2019. URL https://proceedings.mlr.press/v100/jaquier20a.html. 1

[7] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396, 2017. URL https://doi.org/10.1109/ICRA.2017.7989385. 1

[8] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research (IJRR)*, 40(4-5):698–721, 2021. URL https://doi.org/10.1177/0278364920987859. 1

[9] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Found. Trends Robot*, 2(1–2):1–142, 2013. URL https://doi.org/10.1561/2300000021. 1

[10] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 25:328–373, 2013. URL https://doi.org/10.1162/NECO_a_00393. 1

[11] Sylvain Calinon, Florent Guenter, and Aude Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):286–298, 2007. URL https://doi.org/10.1109/TSMCB.2006.886952. 1, 2

[12] Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. Using probabilistic movement primitives in robotics. *Autonomous Robots*, 42:529–551, 2018. URL https://doi.org/10.1007/s10514-017-9648-7. 1

[13] Leonel Rozo and Vedant Dave. Orientation probabilistic movement primitives on Riemannian manifolds. In *Conference on Robot Learning (CoRL)*, pages 373–383, 2021. URL https://proceedings.mlr.press/v164/rozo22a.html. 1

[14] Shikhar Bahl, Mustafa Mukadam, Abhinav Gupta, and Deepak Pathak. Neural dynamic policies for end-to-end sensorimotor learning. In *Neural Information Processing Systems (NeurIPS)*, pages 5058–5069, 2020. URL https://proceedings.neurips.cc/paper/2020/file/354ac345fd8c6d7ef634d9a8e3d47b83-Paper.pdf. 1

[15] Thomas Cederborg, Ming Li, Adrien Baranes, and Pierre-Yves Oudeyer. Incremental local online Gaussian mixture regression for imitation learning of multiple tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 267–274, 2010. URL https://doi.org/10.1109/IROS.2010.5652040. 2

[16] Sylvain Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics*, 9(1):1–29, January 2016. ISSN 1861-2776. URL https://doi.org/10.1007/s11370-015-0187-9. 2

[17] Noémie Jaquier, David Ginsbourger, and Sylvain Calinon. Learning from demonstration with model-based Gaussian process. In *Conference on Robot Learning (CoRL)*, pages 247–257, 2019. URL http://proceedings.mlr.press/v100/jaquier20b.html. 2

[18] N. Jaquier, L. Rozo, D. G. Caldwell, and S. Calinon. Geometry-aware manipulability learning, tracking and transfer. 20(3):624–650, 2021. URL https://doi.org/10.1177/0278364920946815. 2

[19] Jihong Zhu, Michael Gienger, and Jens Kober. Learning task-parameterized skills from few demonstrations. *IEEE Robotics and Automation Letters (RA-L)*, 7(2):4063–4070, 2022. URL https://doi.org/10.1109/LRA.2022.3150013. 2

[20] S. Mohammad Khansari-Zadeh and Aude Billard. Learning stable nonlinear dynamical systems with Gaussian mixture models. *IEEE Transactions on Robotics (T-RO)*, 27(5):943–957, 2011. URL https://doi.org/10.1109/TRO.2011.2159412. 2

[21] Harish Ravichandar, Iman Salehi, and Ashwin Dani. Learning partially contracting dynamical systems from demonstrations. In *Conference on Robot Learning (CoRL)*, pages 369–378, 2017. URL https://proceedings.mlr.press/v78/ravichandar17a.html. 2

[22] Nadia Figueroa and Aude Billard. A physically-consistent Bayesian non-parametric mixture model for dynamical system learning. In *Conference on Robot Learning (CoRL)*, pages 927–946, 2018. URL https://proceedings.mlr.press/v87/figueroa18a.html. 2

[23] Marco Ewerton, Gerhard Neumann, Rudolf Lioutikov, Heni Ben Amor, Jan Peters, and Guilherme Maeda. Learning multiple collaborative tasks with a mixture of interaction primitives. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1535–1542, 2015. URL https://doi.org/10.1109/ICRA.2015.7139393. 2

[24] Leonel Rozo, Sylvain Calinon, Darwin G. Caldwell, Pablo Jiménez, and Carme Torras. Learning physical collaborative robot behaviors from human demonstrations. *IEEE Transactions on Robotics*, 32(3):513–527, 2016. URL https://doi.org/10.1109/TRO.2016.2540623. 2

[25] Maria Kyrarini, Muhammad Abdul Haseeb, Danijela Ristić-Durrant, and Axel Gräser. Robot learning of industrial assembly task via human demonstrations. *Autonomous Robots*, 43:239–257, 2019. URL https://doi.org/10.1007/s10514-018-9725-6. 2

[26] Leonel Rozo, Meng Guo, Andras G. Kupcsik, Marco Todescato, Philipp Schillinger, Markus Giftthaler, Matthias Ochs, Markus Spies, Nicolai Waniek, Patrick Kesper, and Mathias Burger. Learning and sequencing of object-centric manipulation skills for industrial tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9072–9079, 2020. URL https://doi.org/10.1109/IROS45743.2020.9341570. 2

[27] Yun Lin, Shaogang Ren, Matthew Clevenger, and Yu Sun. Learning grasping force from demonstration. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1526–1531, 2012. URL https://doi.org/10.1109/ICRA.2012.6225222. 2

[28] Fares J. Abu-Dakka, Leonel Rozo, and Darwin G. Caldwell. Force-based variable impedance learning for robotic manipulation. *Robotics and Autonomous Systems (RAS)*, 109:156–167, 2018. URL https://doi.org/10.1016/j.robot.2018.07.008. 2

[29] Yanlong Huang, Leonel Rozo, João Silvério, and Darwin G. Caldwell. Kernelized movement primitives. *The International Journal of Robotics Research (IJRR)*, 38(7):833–852, 2019. doi: 10.1177/0278364919846363. URL https://doi.org/10.1177/0278364919846363. 2

[30] Oleg Arenz, Mingjun Zhong, and Gerhard Neumann. Trust-region variational inference with Gaussian mixture models. *Journal of Machine Learning Research (JMLR)*, 21(163):1–60, 2020. URL http://jmlr.org/papers/v21/19-524.html. 2

[31] Iman Nematollahi, Erick Rosete-Beas, Adrian Röfer, Tim Welschehold, Abhinav Valada, and Wolfram Burgard. Robot skill adaptation via soft actor-critic Gaussian mixture models. In *International Conference on Robotics and Automation (ICRA)*, pages 8651–8657, 2022. URL https://doi.org/10.1109/ICRA46639.2022.9811770. 2, 3, 8

[32] Filippo Santambrogio. *Optimal Transport for Applied Mathematicians*. Birkhäuser, Cham, 1 edition, 2015. URL https://doi.org/10.1007/978-3-319-20828-2. 2, 3, 4

[33] Gabriel Peyré and Marco Cuturi. Computational optimal transport: With applications to data science. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019. URL http://dx.doi.org/10.1561/2200000073. 2, 3

[34] Yongxin Chen, Tryphon T. Georgiou, and Allen Tannenbaum. Optimal transport for Gaussian mixture models. *IEEE Access*, 7:6269–6278, 2019. URL https://doi.org/10.1109/ACCESS.2018.2889838. 2, 4, 6

[35] Julie Delon and Agnès Desolneux. A Wasserstein-type distance in the space of Gaussian mixture models. *SIAM Journal on Imaging Sciences*, 13(2):936–970, 2020. URL https://doi.org/10.1137/19M1301047. 2, 4, 6

[36] Pierre H. Richemond and Brendan Maginnis. On Wasserstein reinforcement learning and the Fokker-Planck equation, 2017. URL https://arxiv.org/abs/1712.07185. arXiv:1712.07185. 2, 3, 4, 6

[37] Richard Jordan, David Kinderlehrer, and Felix Otto. The variational formulation of the Fokker–Planck equation. *Siam Journal on Applied Mathematics*, 1996. URL https://doi.org/10.1137/S0036141096303359. 2, 4, 6

[38] Ruiyi Zhang, Changyou Chen, Chunyuan Li, and Lawrence Carin. Policy optimization as Wasserstein gradient flows. In *International Conference on Machine Learning (ICML)*, pages 5737–5746, 2018. URL https://proceedings.mlr.press/v80/zhang18a.html. 2, 3

[39] Petr Mokrov, Alexander Korotin, Lingxiao Li, Aude Genevay, Justin Solomon, and Evgeny Burnaev. Large-scale Wasserstein gradient flows. In *Neural Information Processing Systems (NeurIPS)*, 2021. URL https://openreview.net/forum?id=nlLjIuHsMHp. 2

[40] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, pages 1861–1870, 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html. 3

[41] Jie Ren, Yewen Li, Zihan Ding, Wei Pan, and Hao Dong. Probabilistic mixture-of-experts for efficient deep reinforcement learning, 2021. URL https://arxiv.org/abs/2104.09122. 3, 8, 9

[42] Shijun Wang, Baocheng Zhu, Chen Li, Mingzhe Wu, James Zhang, Wei Chu, and Yuan Qi. Riemannian proximal policy optimization, 2020. URL https://arxiv.org/abs/2005.09195. arXiv:2005.09195. 3

[43] Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998. URL https://doi.org/10.1162/089976698300017746. 3

[44] Sham M Kakade. A natural policy gradient. In *Neural Information Processing Systems (NeurIPS)*, 2001. URL https://proceedings.neurips.cc/paper_files/paper/2001/file/4b86abe48d358ecf194c56c69108433e-Paper.pdf. 3

[45] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347. 3, 5, 8

[46] Alexandre Galashov, Siddhant Jayakumar, Leonard Hasenclever, Dhruva Tirumala, Jonathan Schwarz, Guillaume Desjardins, Wojtek M. Czarnecki, Yee Whye Teh, Razvan Pascanu, and Nicolas Heess. Information asymmetry in KL-regularized RL. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=S1lqMn05Ym. 3

[47] Aldo Pacchiano, Jack Parker-Holder, Yunhao Tang, Krzysztof Choromanski, Anna Choromanska, and Michael Jordan. Learning to score behaviors for guided policy optimization. In *International Conference on Machine Learning (ICML)*, pages 7445–7454, 2020. URL https://proceedings.mlr.press/v119/pacchiano20a.html. 3

[48] Ted Moskovitz, Michael Arbel, Ferenc Huszar, and Arthur Gretton. Efficient Wasserstein natural gradients for reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=OHgnfSrn2jv. 3

[49] Wu Lin, Mark Schmidt, and Mohammad Emtiyaz Khan. Handling the positive-definite constraint in the Bayesian learning rule. In *International Conference on Machine Learning (ICML)*, pages 6116–6126, 2020. URL https://proceedings.mlr.press/v119/lin20d.html. 3

[50] Filippo Santambrogio. Euclidean, metric, and Wasserstein gradient flows: an overview. *Bull. Math. Sci.*, 7: 87–154, 2017. URL https://doi.org/10.1007/s13373-017-0101-1. 3

[51] Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré. *Gradient flows: in metric spaces and in the space of probability measures*. Springer Science & Business Media, 2005. URL https://doi.org/10.1007/b137080. 4, 6

[52] Victor M. Panaretos and Yoav Zemel. *The Wasserstein Space*, pages 37–57. Springer International Publishing, 2020. URL https://doi.org/10.1007/978-3-030-38438-8_2. 4

[53] Zoubin Ghahramani and Michael Jordan. Supervised learning from incomplete data via an EM approach. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1994. URL https://proceedings.neurips.cc/paper/1993/file/f2201f5191c4e92cc5af043eebfd0946-Paper.pdf. 5

[54] Benjamin Eysenbach and Sergey Levine. Maximum entropy RL (provably) solves some robust RL problems. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=PtSAD3caaA2. 5

[55] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, page 1889–1897, 2015. URL https://proceedings.mlr.press/v37/schulman15.html. 5

[56] Fabian Otto, Philipp Becker, Ngo Anh Vien, Hanna Carolin Ziesche, and Gerhard Neumann. Differentiable trust region layers for deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=qYZD-AO1Vn. 5

[57] Felix Otto. The geometry of dissipative evolution equations:: The porus medium equation. *Communications in Partial Differential Equations*, 26(1-2):101–174, 2001. URL https://doi.org/10.1081/PDE-100002243. 6

[58] Yifan Chen and Wuchen Li. Optimal transport natural gradient for statistical manifolds with continuous sample space. *Information Geometry*, 3, 06 2020. URL https://doi.org/10.1007/s41884-020-00028-0. 6, 18

[59] Luigi Malagò, Luigi Montrucchio, and Giovanni Pistone. Wasserstein Riemannian geometry of Gaussian densities. *Information Geometry*, 1:137–179, 2018. URL https://doi.org/10.1007/s41884-018-0014-4. 6, 16, 19

[60] Andi Han, Bamdev Mishra, Pratik Jawanpuria, and Junbin Gao. On Riemannian optimization over positive definite matrices with the Bures-Wasserstein geometry. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL https://openreview.net/forum?id=ZCHxGFmc62a. 6, 7, 16, 19

[61] Nicolas Boumal. *An introduction to optimization on smooth manifolds*. Cambridge University Press, 2023. URL http://www.nicolasboumal.net/book. 6

[62] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 2004. URL https://doi.org/10.1007/BF00992696. 6

[63] Marco Cuturi and Arnaud Doucet. Fast computation of Wasserstein barycenters. In *International Conference on Machine Learning (ICML)*, pages 685–693, 2014. URL https://proceedings.mlr.press/v32/cuturi14.html. 7

[64] James Townsend, Niklas Koep, and Sebastian Weichwald. Pymanopt: A Python toolbox for optimization on manifolds using automatic differentiation. *Journal of Machine Learning Research (JMLR)*, 17(137): 1–5, 2016. URL http://jmlr.org/papers/v17/16-177.html. 7, 19

[65] Rémi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z. Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, Léo Gautheron, Nathalie T.H. Gayraud, Hicham Janati, Alain Rakotomamonjy, Ievgen Redko, Antoine Rolet, Antony Schutz, Vivien Seguy, Danica J. Sutherland, Romain Tavenard, Alexander Tong, and Titouan Vayer. Pot: Python optimal transport. *Journal of Machine Learning Research (JMLR)*, 22(78):1–8, 2021. URL http://jmlr.org/papers/v22/20-451.html. 7

[66] Peter Corke and Jesse Haviland. Not your grandmother's toolbox – the robotics toolbox reinvented for Python. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 11357–11363, 2021. URL https://10.1109/ICRA48506.2021.9561366. 8

[67] Carlo D'Eramo, Davide Tateo, Andrea Bonarini, Marcello Restelli, and Jan Peters. Mushroomrl: Simplifying reinforcement learning research. *Journal of Machine Learning Research (JMLR)*, 22(131):1–5, 2021. URL http://jmlr.org/papers/v22/18-056.html. 8

[68] Francesco Braghin Asad Ali Shahid, Dario Piga and Loris Roveda. Continuous control actions learning and adaptation for robotic manipulation through reinforcement learning. *Autonomous Robots*, 46:483–498, 2022. URL https://doi.org/10.1007/s10514-022-10034-z. 8

[69] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html. 8

[70] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, page 2623–2631, 2019. URL https://doi.org/10.1145/3292500.3330701. 8

[71] James Thornton and Marco Cuturi. Rethinking initialization of the Sinkhorn algorithm, 2022. URL https://arxiv.org/abs/2206.07630. arXiv:2206.07630. 10

[72] Lenaic Chizat, Bernhard Schmitzer, Gabriel Peyré, and François-Xavier Vialard. An interpolating distance between optimal transport and Fisher-Rao, 2015. URL https://arxiv.org/abs/1506.06430. 10

[73] Thomas O. Gallouët and Léonard Monsaingeon. A JKO splitting scheme for Kantorovich–Fisher–Rao gradient flows. *SIAM Journal on Mathematical Analysis*, 49(2):1100–1130, 2017. URL https://doi.org/10.1137/16M106666X. 10

[74] Matthias Liero, Alexander Mielke, and Giuseppe Savaré. Optimal entropy-transport problems and a new Hellinger-Kantorovich distance between positive measures. *Inventiones mathematicae*, 211, 03 2018. URL https://doi.org/10.1007/s00222-017-0759-8. 10

[75] Yuling Yan, Kaizheng Wang, and Philippe Rigollet. Learning Gaussian mixtures using the Wasserstein-Fisher-Rao gradient flow, 2023. URL https://arxiv.org/abs/2301.01766. 10

A Appendix

A.1 Details on Gaussian Mixture Regression (GMR)

In GMR we start from a GMM in state-action space $\pi(\boldsymbol{s}, \boldsymbol{a}) = \sum_{i=1}^{N} \omega_i \mathcal{N}\big([\boldsymbol{s}\, \boldsymbol{a}]^\mathsf{T}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\big)$ from which a policy, i.e. a probability distribution on the action space, can be obtained by conditioning on the state, as follows

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \frac{\pi(\boldsymbol{s}, \boldsymbol{a})}{\int \pi(\boldsymbol{s}, \boldsymbol{a}) \mathrm{d}\boldsymbol{a}}. \tag{15}$$

The resulting conditional distribution is another GMM on the action sapce, with state dependent parameters, given by:

$$\pi(\boldsymbol{a}_t|\boldsymbol{s}_t) = \sum_{i=1}^{N} \omega_i(\boldsymbol{s}_t) \mathcal{N}(\boldsymbol{a}_t; \boldsymbol{\mu}_i^a(s_t), \boldsymbol{\Sigma}_i^a), \quad \text{with} \tag{16}$$

$$\boldsymbol{\mu}_i^a(s_t) = \boldsymbol{\mu}_i^a + \boldsymbol{\Sigma}_i^{as} (\boldsymbol{\Sigma}_i^s)^{-1} \left(\boldsymbol{s}_t - \boldsymbol{\mu}_i^s\right), \tag{17}$$

$$\boldsymbol{\Sigma}_i^a = \boldsymbol{\Sigma}_i^a - \boldsymbol{\Sigma}_i^{as} (\boldsymbol{\Sigma}_i^s)^{-1} \boldsymbol{\Sigma}_i^{sa}, \tag{18}$$

$$\omega_i(\boldsymbol{s}_t) = \frac{\omega_i \mathcal{N}(\boldsymbol{s}_t; \boldsymbol{\mu}_i^s, \boldsymbol{\Sigma}_i^s)}{\sum_k^n \omega_k \mathcal{N}(\boldsymbol{s}_t; \boldsymbol{\mu}_k^s, \boldsymbol{\Sigma}_k^s)}. \tag{19}$$

Note that we have split the GMM parameters $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ into their state and action components according to

$$\boldsymbol{\mu}_i = \begin{pmatrix} \boldsymbol{\mu}_i^s \\ \boldsymbol{\mu}_i^a \end{pmatrix}, \quad \boldsymbol{\Sigma}_i = \begin{pmatrix} \boldsymbol{\Sigma}_i^s & \boldsymbol{\Sigma}_i^{sa} \\ \boldsymbol{\Sigma}_i^{as} & \boldsymbol{\Sigma}_i^a \end{pmatrix}. \tag{20}$$

A.2 Riemannian gradients and retractions

For completeness we give here the explicit expressions of the Riemannian gradients and the retractions used in § 3.1. As the mean vectors are assumed to lie in the Euclidean space, their Riemannian gradients actually coincide with the Euclidean gradients and no retraction is required, so Eq. 12 reduces to the well-known Euclidean gradient descent

$$\hat{\boldsymbol{\mu}}_{k+1} = \hat{\boldsymbol{\mu}}_k + \nabla_{\hat{\boldsymbol{\mu}}} J(\pi_k), \tag{21}$$

where $\nabla_{\hat{\boldsymbol{\mu}}}$ denotes the Euclidean gradient w.r.t. $\hat{\boldsymbol{\mu}}$. For the covariance matrices we use the gradient and retraction w.r.t. the Bures-Wasserstein manifold, taken from [59, 60]. The gradient is given by

$$\mathrm{grad}_{\hat{\boldsymbol{\Sigma}}} J(\pi_k) = 4\{\nabla_{\hat{\boldsymbol{\Sigma}}} J(\pi_k) \hat{\boldsymbol{\Sigma}}\}_S, \tag{22}$$

where again $\nabla_{\hat{\boldsymbol{\Sigma}}}$ denotes the Euclidean gradient w.r.t. $\hat{\boldsymbol{\Sigma}}$ and $\{\boldsymbol{X}\}_S = \frac{(\boldsymbol{X} + \boldsymbol{X}^\mathsf{T})}{2}$.

Furthermore, the retraction is given by

$$\mathrm{R}_{\boldsymbol{\Sigma}_k}\left(\hat{\boldsymbol{X}}\right) = \hat{\boldsymbol{\Sigma}}_k + \hat{\boldsymbol{X}} + \mathcal{L}_{\hat{\boldsymbol{X}}}\left[\hat{\boldsymbol{\Sigma}}_k\right] \hat{\boldsymbol{X}} \mathcal{L}_{\hat{\boldsymbol{X}}}\left[\hat{\boldsymbol{\Sigma}}_k\right], \tag{23}$$

where $\mathcal{L}_{\hat{\boldsymbol{X}}}\left[\hat{\boldsymbol{\Sigma}}_k\right]$ is the Lyapunov operator, defined as the solution to the matrix linear system

$$\mathcal{L}_{\hat{\boldsymbol{X}}}\left[\hat{\boldsymbol{\Sigma}}_k\right] \hat{\boldsymbol{X}} + \hat{\boldsymbol{X}} \mathcal{L}_{\hat{\boldsymbol{X}}}\left[\hat{\boldsymbol{\Sigma}}_k\right] = \hat{\boldsymbol{\Sigma}}_k. \tag{24}$$

A.3 Expressions of the free functional $J(\pi)$ and its Euclidean gradients

For completeness sake, we provide here the explicit expression of the Euclidean gradients for the objective $J(\pi)$ w.r.t. the parameters of the GMM, which are used in the construction of the Riemannian gradients. Using the policy gradient theorem, we obtain the gradient of Eq. 11 w.r.t to a

parameter $\boldsymbol{\xi}$ as follows

$$\nabla_{\boldsymbol{\xi}} J(\pi) = \nabla_{\boldsymbol{\xi}} \int \Pi_t \mathrm{d}\boldsymbol{s}_0 \mathrm{d}\boldsymbol{s}_t \mathrm{d}\boldsymbol{a}_t \rho(\boldsymbol{s}_0)\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)\sum_{t'>t}\gamma^t r(\boldsymbol{s}_{t'},\boldsymbol{a}_{t'}), \tag{25}$$

$$= \mathbb{E}_\tau\left[\sum_t \nabla_{\boldsymbol{\xi}} \log(\pi(\boldsymbol{a}_t|\boldsymbol{s}_t))\sum_{t'>t} r(\boldsymbol{s}_t,\boldsymbol{a}_t)\right],$$

$$= \mathbb{E}_\tau\left[\sum_t \nabla_{\boldsymbol{\xi}} \log\left(\frac{\pi(\boldsymbol{s}_t,\boldsymbol{a}_t)}{\int \mathrm{d}\boldsymbol{a}_t \pi(\boldsymbol{s}_t,\boldsymbol{a}_t)}\right)\sum_{t'>t} r(\boldsymbol{s}_t,\boldsymbol{a}_t)\right],$$

$$= \sum_t \mathbb{E}_\tau\left[\left(\frac{\nabla_{\boldsymbol{\xi}}\pi(\boldsymbol{s}_t,\boldsymbol{a}_t)}{\pi(\boldsymbol{s}_t,\boldsymbol{a}_t)} - \frac{\int \mathrm{d}\boldsymbol{a}_t \nabla_{\boldsymbol{\xi}}\pi(\boldsymbol{s}_t,\boldsymbol{a}_t)}{\int \mathrm{d}\boldsymbol{a}_t \pi(\boldsymbol{s}_t,\boldsymbol{a}_t)}\right)\sum_{t'>t} r(\boldsymbol{s}_t,\boldsymbol{a}_t)\right].$$

In this work, we focus on GMM policies, for which the objective $J(\pi)$ takes the form:

$$J(\pi) = \int \Pi_t \mathrm{d}\boldsymbol{s}_0 \mathrm{d}\boldsymbol{s}_t \mathrm{d}\boldsymbol{a}_t \rho(\boldsymbol{s}_0)\sum_{i=1}^n \omega_i(\boldsymbol{s}_t)\mathcal{N}(\boldsymbol{a}_t;\boldsymbol{\mu}_i(\boldsymbol{s}_t),\boldsymbol{\Sigma}_i(\boldsymbol{s}_t))p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,a_t)\sum_t \gamma^t r(\boldsymbol{s}_t,\boldsymbol{a}_t)$$
$$+\beta\int \mathrm{d}\boldsymbol{a}_t \sum_{i=1}^n \omega_i(\boldsymbol{s}_t)\mathcal{N}(\boldsymbol{a}_t;\mu_i(\boldsymbol{s}_t),\Sigma_i(\boldsymbol{s}_t))p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)$$
$$\log\left(\sum_{i=1}^n \omega_i(s_t)\mathcal{N}(\boldsymbol{a}_t;\mu_i(\boldsymbol{s}_t),\Sigma_i(\boldsymbol{s}_t))p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)\right). \tag{26}$$

By inserting Eq. 26 into Eq. 25 we obtain for the individual parameters of the GMM

$$\nabla_{\boldsymbol{\mu}_l} J(\pi) = \mathbb{E}_\tau\left[\sum_t\left(\frac{\boldsymbol{\omega}_l\mathcal{N}(\boldsymbol{s}_t,\boldsymbol{a}_t;\boldsymbol{\mu}_l,\boldsymbol{\Sigma}_l)\boldsymbol{\Sigma}_l^{-1}\left((\boldsymbol{s}_t,\boldsymbol{a}_t)-\boldsymbol{\mu}_l\right)}{\sum_j \boldsymbol{\omega}_j\mathcal{N}(\boldsymbol{s}_t,\boldsymbol{a}_t;\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)}\right.\right. \tag{27}$$
$$\left.\left.-\frac{\boldsymbol{\omega}_l\int \mathrm{d}\boldsymbol{a}\mathcal{N}(\boldsymbol{s}_t,\boldsymbol{a}_t;\boldsymbol{\mu}_l,\boldsymbol{\Sigma}_l)\boldsymbol{\Sigma}_l^{-1}\left((\boldsymbol{s}_t,\boldsymbol{a}_t)-\boldsymbol{\mu}_l\right)}{\sum_j \boldsymbol{\omega}_j\int \mathrm{d}\boldsymbol{a}\mathcal{N}(\boldsymbol{s}_t,\boldsymbol{a}_t;\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)}\right)\sum_{t'>t} r(\boldsymbol{s}_t,\boldsymbol{a}_t)\right],$$

$$= \mathbb{E}_\tau\left[\sum_t\left(\frac{\boldsymbol{\omega}_l\mathcal{N}(\boldsymbol{s}_t,\boldsymbol{a}_t;\boldsymbol{\mu}_l,\boldsymbol{\Sigma}_l)\boldsymbol{\Sigma}_l^{-1}\left((\boldsymbol{s}_t,\boldsymbol{a}_t)-\boldsymbol{\mu}_l\right)}{\sum_j \boldsymbol{\omega}_j\mathcal{N}(\boldsymbol{s}_t,\boldsymbol{a}_t;\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)}\right.\right. \tag{28}$$
$$\left.\left.-\delta_{\boldsymbol{s}}\frac{\boldsymbol{\omega}_l\mathcal{N}(\boldsymbol{s}_t;\boldsymbol{\mu}_{l,s}\boldsymbol{\Sigma}_{l,ss})\boldsymbol{\Sigma}_{l,ss}^{-1}\left(\boldsymbol{s}_t-\boldsymbol{\mu}_{l,s}\right)}{\sum_j \boldsymbol{\omega}_j\mathcal{N}(\boldsymbol{s}_t;\boldsymbol{\mu}_{j,s},\boldsymbol{\Sigma}_{j,ss})}\right)\sum_{t'>t} r(\boldsymbol{s}_t,\boldsymbol{a}_t)\right],$$

$$= \mathbb{E}_\tau\left[\sum_t\left(\zeta_{l,\boldsymbol{s}_t,\boldsymbol{a}_t}\boldsymbol{\Sigma}_l^{-1}\left((\boldsymbol{s}_t,\boldsymbol{a}_t)-\boldsymbol{\mu}_l\right)-\delta_s\zeta_{l,\boldsymbol{s}_t}\boldsymbol{\Sigma}_{l,ss}^{-1}\left(\boldsymbol{s}_t-\boldsymbol{\mu}_{l,s}\right)\right)\sum_{t'>t} r(\boldsymbol{s}_t,\boldsymbol{a}_t)\right].$$

Here $\delta_{\boldsymbol{s}} \in \{0,1\}$ indicates which terms the gradient acts on. In this case, the gradient act on the state components and it is absent for the action dimensions.

$$\nabla_{\boldsymbol{\Sigma}_l} J(\pi) = \mathbb{E}_\tau\left[\sum_t\left(-\frac{1}{2}\frac{\boldsymbol{\omega}_l\mathcal{N}(\boldsymbol{s}_t,\boldsymbol{a}_t;\boldsymbol{\mu}_l,\boldsymbol{\Sigma}_l)\boldsymbol{\Sigma}_l^{-1}\left(1-\left((\boldsymbol{s}_t,\boldsymbol{a}_t)-\boldsymbol{\mu}_l\right)\left((\boldsymbol{s}_t,\boldsymbol{a}_t)-\boldsymbol{\mu}_l\right)^\mathsf{T}\boldsymbol{\Sigma}_l^{-1}\right)}{\sum_j \boldsymbol{\omega}_j\mathcal{N}(\boldsymbol{s}_t,\boldsymbol{a}_t;\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)}\right.\right. \tag{29}$$
$$\left.\left.+\frac{1}{2}\frac{\boldsymbol{\omega}_l\int \mathrm{d}\boldsymbol{a}\mathcal{N}(\boldsymbol{s}_t,\boldsymbol{a}_t;\boldsymbol{\mu}_l,\boldsymbol{\Sigma}_l)\boldsymbol{\Sigma}_l^{-1}\left(1-\left((\boldsymbol{s}_t,\boldsymbol{a}_t)-\boldsymbol{\mu}_l\right)\left((\boldsymbol{s}_t,\boldsymbol{a}_t)-\boldsymbol{\mu}_l\right)^\mathsf{T}\boldsymbol{\Sigma}_l^{-1}\right)}{\sum_j \boldsymbol{\omega}_j\int \mathrm{d}a\mathcal{N}(\boldsymbol{s}_t,\boldsymbol{a}_t;\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)}\right)\sum_{t'>t} r(\boldsymbol{s}_t,\boldsymbol{a}_t)\right],$$

$$= \mathbb{E}_\tau\left[\sum_t\left(-\frac{\zeta_{l,\boldsymbol{s}_t,\boldsymbol{a}_t}}{2}\boldsymbol{\Sigma}_l^{-1}\left(1-\left((\boldsymbol{s}_t,\boldsymbol{a}_t)-\boldsymbol{\mu}_l\right)\left((\boldsymbol{s}_t,\boldsymbol{a}_t)-\boldsymbol{\mu}_l\right)^\mathsf{T}\boldsymbol{\Sigma}_l^{-1}\right)\right.\right.$$
$$\left.\left.+\delta_s\frac{\zeta_{l,\boldsymbol{s}_t}}{2}\boldsymbol{\Sigma}_{l,s}^{-1}\left(1-\left(\boldsymbol{s}_t-\boldsymbol{\mu}_{l,s}\right)\left(\boldsymbol{s}_t-\boldsymbol{\mu}_{l,s}\right)^\mathsf{T}\boldsymbol{\Sigma}_{l,s}^{-1}\right)\right)\sum_{t'>t} r(\boldsymbol{s}_t,\boldsymbol{a}_t)\right].$$

$$\nabla \boldsymbol{\omega}_l J(\pi) = \mathbb{E}_\tau \left[\sum_t \left(\frac{\mathcal{N}(\boldsymbol{s}_t, \boldsymbol{a}_t; \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}{\sum_j \boldsymbol{\omega}_j \mathcal{N}(\boldsymbol{s}_t, \boldsymbol{a}_t; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} - \frac{\int \mathrm{d}\boldsymbol{a} \mathcal{N}(\boldsymbol{s}_t, \boldsymbol{a}_t; \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}{\sum_j \boldsymbol{\omega}_j \int \mathrm{d}\boldsymbol{a} \mathcal{N}(\boldsymbol{s}_t, \boldsymbol{a}_t; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \right) \sum_{t'>t} r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right] \tag{30}$$

$$= \mathbb{E}_\tau \left[\sum_t \frac{(\zeta_{l,\boldsymbol{s}_t,\boldsymbol{a}_t} - \zeta_{l,\boldsymbol{s}_t})}{\boldsymbol{\omega}_l} \sum_{t'>t} r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]. \tag{31}$$

Note that we introduced the responsibilities $\zeta_{l,\boldsymbol{s}_t,\boldsymbol{a}_t}$ and $\zeta_{l,\boldsymbol{s}_t}$, which are defined as follows

$$\zeta_{l,\boldsymbol{s}_t,\boldsymbol{a}_t} = \frac{\boldsymbol{\omega}_l \mathcal{N}(\boldsymbol{s}_t, \boldsymbol{a}_t; \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}{\sum_j \boldsymbol{\omega}_j \mathcal{N}(\boldsymbol{s}_t, \boldsymbol{a}_t; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}, \quad \text{and} \tag{32}$$

$$\zeta_{l,\boldsymbol{s}_t} = \frac{\boldsymbol{\omega}_l \int \mathrm{d}\boldsymbol{a} \mathcal{N}(\boldsymbol{s}_t, \boldsymbol{a}_t; \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}{\sum_j \boldsymbol{\omega}_j \int \mathrm{d}\boldsymbol{a} \mathcal{N}(\boldsymbol{s}_t, \boldsymbol{a}_t; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} = \frac{\boldsymbol{\omega}_l \mathcal{N}(\boldsymbol{s}_t; \boldsymbol{\mu}_{l,s}, \boldsymbol{\Sigma}_{l,ss})}{\sum_j \boldsymbol{\omega}_j \mathcal{N}(\boldsymbol{s}_t; \boldsymbol{\mu}_{j,s}, \boldsymbol{\Sigma}_{j,ss})}. \tag{33}$$

A.4 Relation between forward and backward discretization in the Bures-Wasserstein metric

In this section we outline the relation between the implicit and explicit optimization schema w.r.t. the Bures-Wasserstein metric, which is leveraged to formulate our policy optimization in § 3. We closely follow [58]. For the sake of simplicity, we group the Gaussian parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ into a single parameter vector $\boldsymbol{\theta}$. Furthermore, we restrict our explanation to a single Gaussian component, which is possible without loosing generality, as each of the N components live in its own manifold $\mathbb{R}^d \times \mathcal{S}^d_{++}$.

The Riemannian gradient w.r.t the Gaussian parameters $\boldsymbol{\theta}$, $\mathrm{grad}_{\boldsymbol{\theta}} J(\pi(\boldsymbol{\theta}))$, satisfies by definition

$$g_{\boldsymbol{\theta}}(\mathrm{grad}_{\boldsymbol{\theta}} J(\pi(\boldsymbol{\theta}), \boldsymbol{\xi}) = \nabla_{\boldsymbol{\theta}} J(\pi(\boldsymbol{\theta})) \cdot \boldsymbol{\xi}, \tag{34}$$

where $\nabla_{\boldsymbol{\theta}}$ denotes the Euclidean gradient, $\boldsymbol{\xi}$ is an arbitrary vector on the tangent space $\mathcal{T}_{\boldsymbol{\theta}}\mathcal{M}$, and $g_{\boldsymbol{\theta}}$ is the Riemannian metric tensor, defining the inner product on $\mathcal{T}_{\boldsymbol{\theta}}\mathcal{M}$. The Riemannian metric $g_{\boldsymbol{\theta}}$ can be written as

$$g_{\boldsymbol{\theta}}(\boldsymbol{\zeta}, \boldsymbol{\xi}) = \boldsymbol{\zeta}^\mathsf{T} G_W(\boldsymbol{\theta}) \boldsymbol{\xi}, \tag{35}$$

with two arbitrary tangent vectors $\boldsymbol{\zeta}$, $\boldsymbol{\xi}$, and $G_W(\boldsymbol{\theta})$ being a positive definite matrix. Moreover, note that the Wasserstein distance $W_2^2\big(\mathcal{N}(\boldsymbol{\theta}), \mathcal{N}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta})\big)$, where $\Delta\boldsymbol{\theta}$ denotes a small perturbation in the Gaussian parameters $\boldsymbol{\theta}$, can be expressed as

$$W_2^2\big(\mathcal{N}(\boldsymbol{\theta}), \mathcal{N}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta})\big) = \frac{1}{2}\big(\Delta\boldsymbol{\theta}^\mathsf{T}\big) G_W(\boldsymbol{\theta}) \big(\Delta\boldsymbol{\theta}\big) + O\big((\boldsymbol{\Delta\theta})^2\big), \tag{36}$$

for $\Delta\boldsymbol{\theta} \to 0$. Similarly, we can approximate the objective evaluated at $J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta})$ via the Taylor theorem as

$$J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \cdot \Delta\boldsymbol{\theta} + O((\Delta\boldsymbol{\theta})^2). \tag{37}$$

With this, we can approximate

$$\boldsymbol{\theta}_{k+1} = \arg\min_{\boldsymbol{\theta}} \left(\frac{W_2^2(\pi(\boldsymbol{\theta}), \pi(\boldsymbol{\theta}_k))}{2\tau} - J(\pi(\boldsymbol{\theta})) \right), \tag{38}$$

$$\approx \arg\min_{\boldsymbol{\theta}} \left(\frac{(\boldsymbol{\theta} - \boldsymbol{\theta_k})^\mathsf{T} G_W (\boldsymbol{\theta} - \boldsymbol{\theta_k})}{2\tau} - \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \cdot (\boldsymbol{\theta} - \boldsymbol{\theta_k}) \right), \tag{39}$$

from which we obtain the update equation for $\boldsymbol{\theta}$ as follows

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \tau G_W(\boldsymbol{\theta}_k)^{-1} \nabla_{\boldsymbol{\theta}} J(\pi(\boldsymbol{\theta}_k)), \tag{40}$$

which corresponds to the Wasserstein natural gradient with respect to the Bures-Wasserstein metric. Note that Eq. 40 in turn corresponds to an approximation of the exact Riemannian gradient descent

$$\boldsymbol{\theta}_{k+1} = \mathrm{R}_{\boldsymbol{\theta}_k} \left(\tau \cdot \mathrm{grad}_{\boldsymbol{\theta}} J(\pi(\boldsymbol{\theta}_k)) \right). \tag{41}$$

This approximation can be obtained by considering a first-order approximation of the geodesic on the BW manifold. As the exponential map (a.k.a. the retraction) is defined via the geodesic, the retraction operator in Eq. 41 turns into a simple addition operation under a first-order approximation, leading to Eq. 40. Notice that such approximation does not guarantee that the updated parameters $\boldsymbol{\theta}$ stay on the manifold, except for the cases in which $\boldsymbol{\theta} \in \mathbb{R}^d$. This means that the positive-definite constraints arising from the covariance matrices can be easily violated when using natural gradient approaches as they do not guarantee that the updates stay on the underlying manifold. However, it is worth noting that the use of an inexact Riemannian gradient descent is often motivated by the difficulty of computing the exponential map (or geodesic) necessary to calculate the exact Riemannian update. In our case, we leverage the retraction and Riemannian gradients of [59, 60], which allow us to apply the exact Riemannian gradient descent of 41. This avoids to rely on first-order approximations and in turn we can guarantee that the updates of the Gaussian distribution parameters always lie on on the product manifold $\left(\mathbb{R}^d \times \mathcal{S}^d_{++}\right)^N$.

A.5 Additional details on the implementation

We extended the Pymanopt [64] by adding a custom line-search routine that accounts for a constraint on the Wasserstein distance between the old and the optimized GMMs. The details of this line-search can be found in Algorithm 2.

Algorithm 2 Constrained line-search. The constraint function $c(x_0.\cdot)$ is arbitrary in general. We use the L^2-Wasserstein distance between two points on the manifold of GMMs as constraint.

Input: point $\boldsymbol{x_0}$ on the manifold, descent direction $\boldsymbol{d}$, initial step size λ_0, decrement α, constraint $c(\boldsymbol{x_0}, \cdot)$, maximum allowed value for constraint c_{max}, minimum step size λ_{min}
Output: step size s, updated point on manifold $\boldsymbol{x}$

1: $\boldsymbol{x} = \boldsymbol{x_0} + \lambda_0 \cdot \boldsymbol{d}$
   $\lambda = \lambda_0$
2: **while** $c(\boldsymbol{x_0}, \boldsymbol{x}) > c_{\text{max}}$ and $\lambda > \lambda_{\text{min}}$ **do**
3: decrease step size: $\lambda = \alpha \cdot \lambda$
   update point on manifold: $\boldsymbol{x} = \boldsymbol{x_0} + \lambda \cdot \boldsymbol{d}$
4: **end while**
5: **if** $\lambda < \lambda_{\text{min}}$ **then**
6: return λ_0, $\boldsymbol{x_0}$
7: **else**
8: return λ, $\boldsymbol{x}$
9: **end if**

A.6 Additional details on experiments

A.6.1 Additional results

Fig. 5 shows the convergence curves for the two baselines as in Fig. 3 of the main paper, however, we extended the horizontal axis up to the maximum number of environment steps used for training.

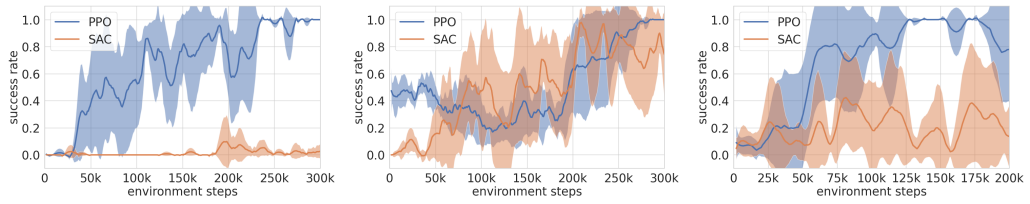


Figure 5: The success rate of the two baselines on the reaching task (*left*), the collision-avoidance task (*middle*) and the multiple-goal task (*right*). The shaded area indicates the standard deviation over 5 runs.

Fig. 6 shows the variance of the success rate for the three methods at their time step of convergence for all three robotic tasks. Concerning SAC, which did not converge after the maximum number of

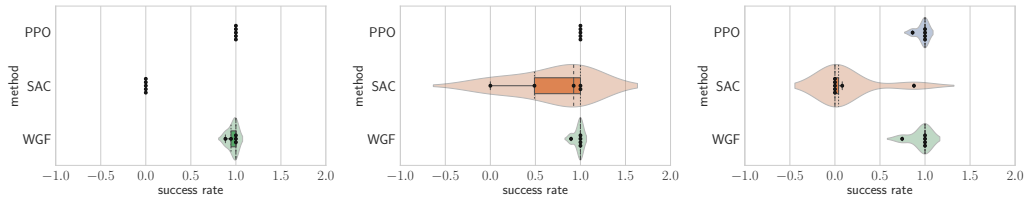


Figure 6: Variance of the success rate over the 5 runs for our method (WGF) and the two baselines on the reaching task (*left*), the collision avoidance task (*middle*) and the multiple-goal task (*right*). The violine plots are overlaid with box plots, quartile lines and a swarm plot, where dots indicate the success rates of individual runs. The time steps at which we determined the variance are for PPO, SAC and WGF for the three tasks from left to right: $(280000, 400000, 80000)$, $(275000, 300000, 90000)$, $(130000, 200000, 95000)$.

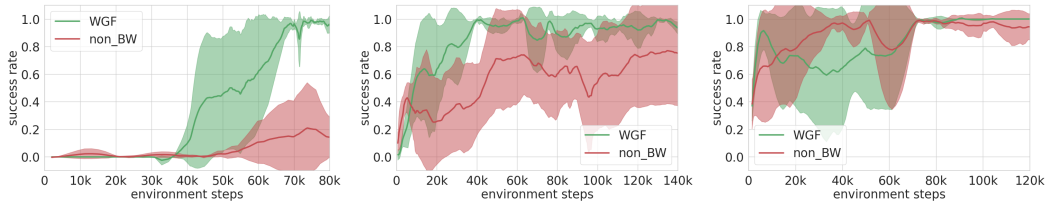


Figure 7: The success rate of our method and an ablated version, not using the Bures-Wasserstein formulation for the reaching task (*left*), the collision-avoidance task (*middle*) and the multiple-goal task (*right*). The shaded area indicates the standard deviation over 5 runs.

environment steps used for training, we chose the last time step. Specifically, we chose the following time steps for PPO, SAC and WGF, respectively: reaching task $(280000, 400000, 80000)$, collision avoidance task $(275000, 300000, 90000)$, multiple goal task $(130000, 200000, 95000)$. These plots show that PPO may also reach low-variance success rate over the five runs at the time step of convergence, at the cost of a prohibitively large number of steps. SAC showed huge variance in all tasks, apart from the reaching task, where all runs collapsed to a success rate of 0.

A.6.2 Ablation study

In order to assess the influence of leveraging a Riemannian optimization approach on the Bures-Wasserstein manifold, we conducted an ablation of our method by eliminating the Riemannian formulation. Instead of the explicit Euler scheme update in Eq. 12, which corresponds to Riemannian gradient descent w.r.t. the Bures-Wasserstein metric, we use the implicit Euler scheme

$$\hat{\boldsymbol{\mu}}_{k+1} = \arg\min_{\hat{\boldsymbol{\mu}}} \left(\frac{W_2^2(\pi_k(\hat{\boldsymbol{\mu}}), \pi_k)}{2\tau} - J(\pi_k(\hat{\boldsymbol{\mu}})) \right), \tag{42}$$

$$\hat{\boldsymbol{\Sigma}}_{k+1} = \arg\min_{\hat{\boldsymbol{\Sigma}}} \left(\frac{W_2^2(\pi_k(\hat{\boldsymbol{\Sigma}}), \pi_k)}{2\tau} - J(\pi_k(\hat{\boldsymbol{\Sigma}})) \right). \tag{43}$$

To guarantee that the updated covariance matrices do not leave the manifold of symmetric positive definite matrices, we parameterize them in terms of Cholesky factors. The results obtained with this non-Riemannian version of our method are shown in Fig. 7 in direct comparison to our method and Fig. 8 for an extended range.

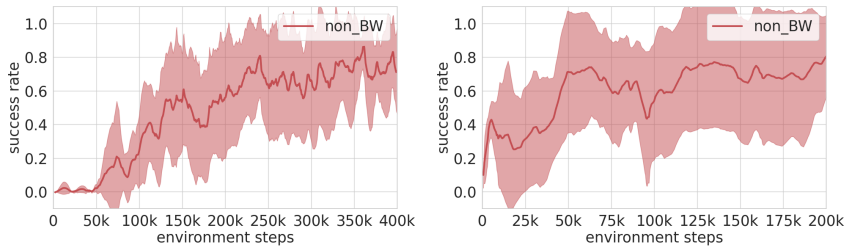


Figure 8: Extended plot of the success rate of and ablated version of our method, not using the Bures-Wasserstein-based formulation for the reaching task (*left*), the collision-avoidance task (*middle*) and the multiple-goal task (*right*). The shaded area indicates the standard deviation over 5 runs.

The results clearly show that the non-Riemannian method struggles to reach a success rate of 1 for the reaching task and the collision-avoidance task. Furthermore, we observe a high variance over different runs in the same settings (see Fig. 9 and Fig. 10). We attribute this to the fact that the our method takes exact gradient steps in the direction of steepest descent w.r.t. the underlying BW metric, whereas the implicit scheme only approximates this direction. For this reason the non-Riemannian method is much more noisy, which in turn leads to the aforementioned high variance. Nevertheless, the multiple-goal task constitutes an exception. Here we observed a similar performance for our approach and the ablated method. The reason for this is that the optimization of this task is mainly dominated by the weight updates, which are identical for both methods. This result is therefore expected and confirms that correctness of our ablation strategy.

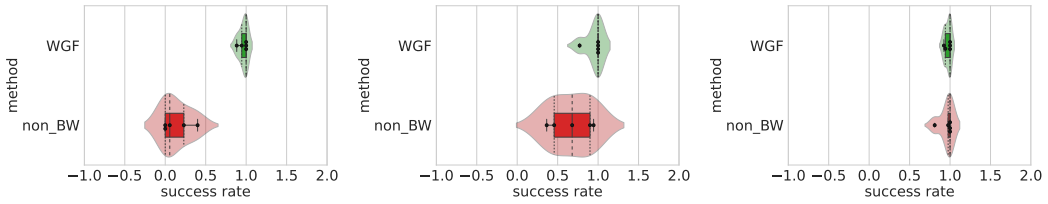


Figure 9: Variance of the success rate over 5 runs for our method (WGF) and the ablated method (non-BW) on the reaching task (*left*), the collision avoidance task (*middle*) and the multiple-goal task (*right*). The violine plots are overlaid with box plots, quartile lines and a swarm plot, where dots indicate the success rates of individual runs. The time steps at which we determined the variance are $80000, 90000, 85000$.

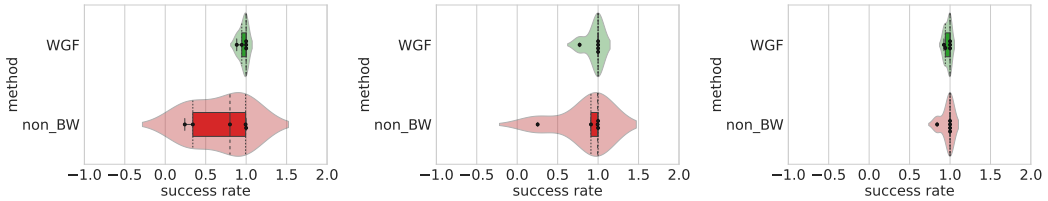


Figure 10: Variance of the success rate over 5 runs for our method (WGF) and the ablated method (non-BW) on the reaching task (*left*), the collision avoidance task (*middle*) and the multiple-goal task (*right*). The violine plots are overlaid with box plots, quartile lines and a swarm plot, where dots indicate the success rates of individual runs. The time steps at which we determined the variance are $(80000, 400000), (90000, 200000), (85000, 90000)$.

A.6.3 Additional experiments with 7-DoF robotic manipulator

We carried out two additional experiments to show that our method can be employed on tasks performed by off-the-shelf robotic manipulators (e.g. a 7-DoF Franka Emika Panda robot). Specifically, we extended the collision-avoidance task described in § 4 to a 3D environment, where the state and action depend on the task space representation. The first task was represented in the robot operational space, i.e., the state $\boldsymbol{s} = \boldsymbol{x} \in \mathbb{R}^3$ and the action $\boldsymbol{a} = \dot{\boldsymbol{x}} \in \mathbb{R}^3$. The second task was represented in the robot joint space, consequently the state $\boldsymbol{s} = \boldsymbol{q} \in \mathbb{R}^7$ and the action $\boldsymbol{a} = \dot{\boldsymbol{q}} \in \mathbb{R}^7$. This experiment was aimed at assessing the capabilities of our approach to adapt robot motion policies in state-action spaces of higher dimensions. The initial 3-components GMM policy was trained using 10 human demonstrations featuring linear reaching 3D trajectories. For policy optimization, we used a sparse reward defined as a function of the position error between the robot end-effector position and the target at the end of the rollout. Moreover, two sparse penalty terms were added to punish collision with obstacles and divergent trajectories.

Similarly to the planar task reported in the main paper, we tested whether our method was able to adapt a trajectory tracking skill in order to avoid collisions with newly added obstacles. This means that the robot end-effector needed to pass through a narrow path between two spherical obstacles. For the operational space representation, the robot end-effector pose was controlled using a full-pose Cartesian velocity controller at a frequency of 100Hz, where the end-effector orientation was kept constant. For the joint space representation, the robot joint configuration was controlled using a joint velocity controller at a frequency of 100Hz. Figure 11 shows the results for 3D operational space representation, where our method reached a success rate of 1.0 very quickly, taking approximately

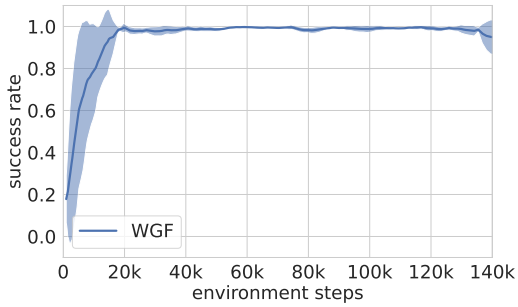


Figure 11: The success rate of our method applied to the 3D narrow-path task performed by the 7-DoF Panda robotic manipulator. The shaded area indicates the standard deviation over 5 runs.

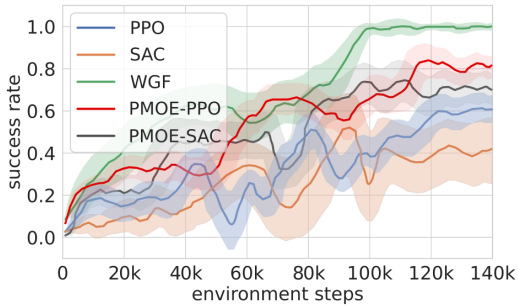


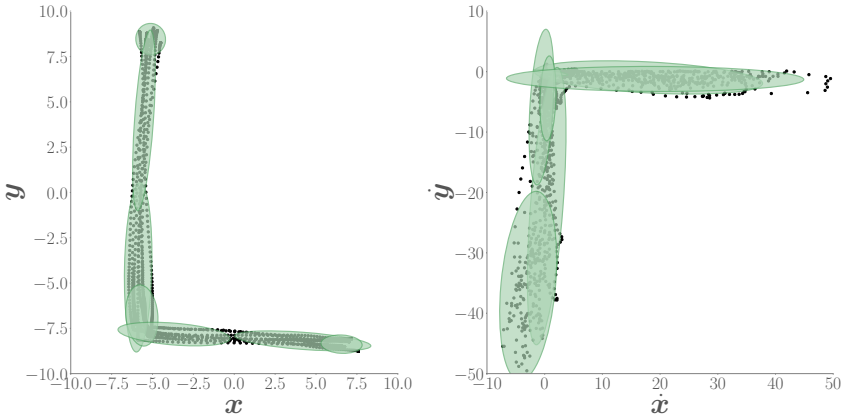
Figure 12: The success rate of our method (WFG) and the baselines on a 3D narrow-path task performed by a simulated 7-DoF Panda robotic manipulator. The task is represented in the robot joint space, therefore the state $\boldsymbol{s} = \boldsymbol{q} \in \mathbb{R}^7$, and the action $\boldsymbol{a} = \dot{\boldsymbol{q}} \in \mathbb{R}^7$, correspond to the joint position and velocity, respectively. The shaded area indicates the standard deviation over 15 runs.

20000 environment steps[3]. Moreover, the solution variance of our method was also very low, which is consistent with our observations concerning the performance of our policy optimization on the three planar tasks analyzed in the main paper. Figure 12 shows the results of the collision avoidance task using the joint space representation. As observed, our approach was able to adapt the robot motion policy so that the robot end-effector safely passes through a narrow path defined by two spherical obstacles (the narrow-path task description is provided in Sec. 4.1). It is evident that our approach outperformed all the baselines in this simulated robotic task, providing evidence that our approach scales and it is able to adapt robot motion policies in higher-dimensional tasks.

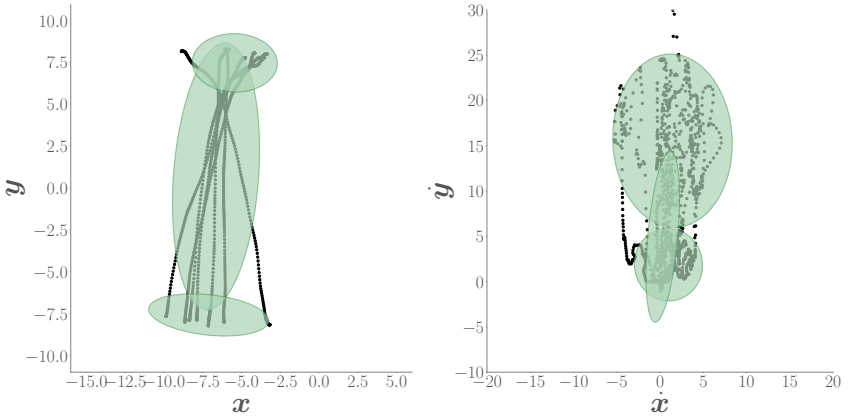
A.6.4 Initial GMM policies

For the sake of completeness, Fig. 13 provides 2D projections of the initial GMM policies learned from demonstrations for the three robotic settings considered in the main paper: the reaching motion skill, the collision-free trajectory tracking, and the multiple-goal task. Figure 13 also provides the demonstration data used to train the initial policies. Note that these models are then adapted according to the policy optimization approach introduced in § 3.2.

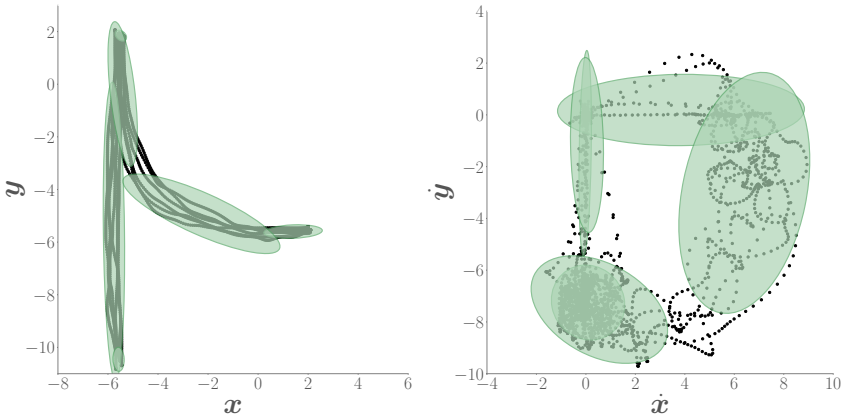
[3]As the baselines underperformed in the 2D case, they were not tested in this specific setting.



(a) Reaching motion skill



(b) Collision-free trajectory tracking



(c) Multiple-goal task

Figure 13: Green Gaussian components (⬭) represent the initial GMM policy learned from demonstrations, projected on the Cartesian position (*left*) and velocity (*left*) spaces. The recorded position and velocity data are depicted as black dots (•).