Noname manuscript No.
(will be inserted by the editor)

Learning motions from demonstrations and rewards with time-invariant dynamical systems based policies

Joel Rey · Klas Kronander · Farbod Farshidian · Jonas Buchli · Aude Billard

the date of receipt and acceptance should be inserted later

Abstract An important challenge when using Reinforcement Learning for learning motions in robotics is the choice of parameterization for the policy. We use Gaussian Mixture Regression to extract a parameterization with relevant non-linear features from a set of demonstrations of a motion following the paradigm of Learning from Demonstration. The resulting parameterization takes the form of a non-linear time-invariant dynamical system (DS). We use this time-invariant DS as a parameterized policy for a variant of the PI^2 policy search algorithm. This paper contributes by adapting PI^2 for our time-invariant motion representation. We introduce two novel parameter exploration schemes that can be used to 1) sample model parameters to achieve a uniform exploration in state space and 2) explore while ensuring stability of the resulting motion model. Additionally, a state dependent stiffness profile is learned simultaneously to the reference trajectory and both are used together in a variable impedance control architecture. This learning architecture is validated in a hardware experiment consisting of a digging task using a KUKA LWR platform.

1 Introduction

A limiting factor in the spread of the use of robotic systems in a wider variety of applications is the tediousness of traditional methods for programming robots. Every time a robot is used for a new task, many hours of work by trained people are required to endow the robot with a new skill. These problems can be addressed by endowing the robot to learn tasks instead of being explicitly programmed in the traditional sense. In Learning from Demonstration (LfD), the goal is to extract task-relevant information from a set of demonstrations. The demonstrations are used to optimize a task model which can subsequently be used for autonomous task execution by the robot [1,2]. Alternatively, with Reinforcement Learning (RL) the robot is expected to learn on its own by executing trials of the task and evaluating these with a cost function that compactly describes the task at hand [3,4]. These two paradigms are complementary and often used in conjunction, e.g. by using LfD as a means to find a good initial task model which is subsequently refined by RL [5,6].

The choice of motion representation is crucial in both LfD and RL. Dynamical Systems (DS) have emerged as a general and efficient method for compactly representing a task. In a DS representation, the state variables are related to their derivatives through a set of parameterized differential equations. Key properties such as global convergence to a target point [7] or cyclic motion [8] can be ensured by a proper choice of parameterization and constraints.

A limitation of current state-of-the art RL methods for learning DS models arises from the chosen parameterization. Models that are to be flexible enough to capture complex motions rely on non-linear features of the state variables. The selection of features play a critical role in the representational power of the policy and hence the quality of the solution. In DS representations, nonlinear features are typically built using Gaussian basis functions following two different approaches. The first approach is to use a uni-dimensional input for the basis functions that is either time or an

Joel Rey, Klas Kronander and Aude Billard are with the Learning Algorithms ans Systems Laboratory (LASA), EPFL, Switzerland

Farbod Farshidian and Jonas Buchli are with the Agile and Dexterous Robotics Lab (ADRL), ETHZ, Switzerland

auxiliary phase variable [9–11,8]. It is then sufficient to spread the basis function regularly along this single dimension. The second approach, which is less common, is to use the state variables of the system as input to the basis function [12,7,13]. Those two approaches have different limitations. The first results in a behavior which ultimately is time-dependent unless a mechanism to control the phase variable based on the robot state is introduced. The second approach results in a time-invariant DS which leads to benefits such as increased robustness [12] [7], but requires an appropriate placement of Gaussians in the state space which can be difficult to obtain. In this work, we follow the second approach. For placement of the nonlinear features and for initialization of the motion policy, we employ demonstrations of the task modeled with Gaussian Mixture Model (GMM). An time-invariant DS model is then obtained through Gaussian Mixture Regression, as in [12], resulting in a motion representation with the basis functions automatically placed in relevant areas of the dataset. Subsequently, we use an adaptation of the state-of-the-art RL algorithm PI^2 to improve the model learned from demonstrations. Hence, our work builds on [12] in our choice of motion representation, on [14], [15] and [16] in the choice of learning algorithm. Our main contribution is to combine these two independent sets of research (namely on time-invariant DS encoding of motion and use of PI2 to improve policies). Precisely, we extend [12] by adding RL as scheme to improve the policy after training from learning from demonstration, and we extend [16] to be used for time-invariant policies, in place of time-dependent encoding policy as was done in the past.

For execution on a physical system, DS task models are typically used as an online trajectory generators for a position controller or impedance controller. In the latter case, the behavior of the robot is determined not only by the DS but also strongly by the impedance parameters. In a learning setting, it hence makes sense to include the tunable impedance parameters (usually limited to the stiffness matrix) as a learnable object. In this paper, we exploit the state-dependent stiffness representation of [17] to ensure synchronization with our motion representation. The stiffness is learned simultaneously with the nominal motion, as was previously introduced for time-dependent motion and stiffness in [16].

Exploration is an important part of any RL system. In this paper, we propose two parameter sampling techniques that are tailored to our parameterization. The first is aimed at mapping locally uniform exploration in velocity space to the associated exploration in parameter space. A major difficulty in applying learning to DS motion representations is that parameter perturbations can yield unstable models. To deal with this issue, we use results from [7] to derive a second exploration strategy with stability guarantee. This is particularly important in tasks that have a fixed target point that should be maintained during exploration.

In summary, the main contributions of this paper are:

1. We show how demonstrations of a task can be used to extract both non-linear features (i.e. basis functions) and initial parameters for a parameterized policy based on time-invariant non-linear DS (Section 3.1).
2. We show that the policy search algorithm PI^2 can be adapted to refine this type of time-invariant policy (Section 3.2).
3. We present two different ways of shaping the exploration noise to obtain
   (a) an homogeneous and isotropic exploration in task space (Section 4.1)
   (b) preservation of the stability properties of the initial policy (Section 4.2)
4. We present a method for learning a state dependent stiffness simultaneously to the reference trajectory, thereby obtaining a variable impedance controller (Section 5).

2 State of the art

2.1 Representing motions for control in robotics

In robot control, encoding motions as Dynamical Systems (DS) has proven to be convenient because it can provide a velocity or acceleration profile in addition to the geometrical information. Furthermore, a DS can generate different trajectories for different starting conditions, which can provide generalization as well as the ability to encode how the robot should react to a perturbation.

The most widely used DS for encoding robotic motions are time-dependent DS. Most of those DS are linear in the state variables and use non-linear basis functions with the time or a phase auxiliary state as input [8]. They have convenient properties, such as the possibility to allow non-constrained learning while ensuring qualitative behavior such as global stability at an attractor point or convergence to a limit cycle. There have been numerous successful demonstrations of learning from demonstration and reinforcement learning in such motion representations [18,19,5] learning motions for a large variety of tasks.

Time-invariant DS formulations can represent richer behaviors because different motions can be learned in different parts of the state space [12]. They can encode a motion in a way that allows for on-the-fly adaptation to spatial and temporal perturbations. Such an intrinsic robustness is desirable for robots operating in uncertain environments. Several time-invariant DS representations have been introduced, along with algorithms for learning them [7,13]. The main drawback of these models compared to time- or phase-dependent models is that stability is generally more difficult to guarantee and can easily be lost during learning. Different methods for obtaining stable non-linear DS have been proposed. In [20], an approach combining Gaussian Mixture Regression (GMR) [21] and Hidden Markov Model (HMM) is used to compute a reference velocity based on the position and sequential information. The system is stabilized by a spring and damper system that attracts trajectories toward the regions where demonstrations have been provided. In [7], the Stable Estimator of Dynamical Systems (SEDS) is presented. This method is based on GMR, with additional constraints to ensure global asymptotic convergence of the DS. In [13], a method based on Extreme Learning Machines for learning a DS that is stable in a given workspace is presented. Other approaches represent the DS in a form that makes it impossible to alter the stability properties during learning [22]. All these DS can be learned from demonstrations. In this paper we use the the stability conditions originally proposed in [23]. These conditions were used as constraints for an optimization problem in [23]. In contrast, we employ these stability conditions to derive a novel parameter sampling strategy that can be used to guarantee that all explored policies remain stable.

2.2 Policy search methods for learning motions from experience

Policy search methods have become popular for RL in robotics because they adapt well to problems with large continuous state and action spaces. Examples include [24], where the Natural Actor Critic (NAC) algorithm is introduced and applied to the task of learning a baseball swing with a robot arm. As policy, a DMP [8] is used, which encodes a reference trajectory for each joint as a time dependent DS. In [10], NAC is used to adapt a different kind of policy, based on a time conditioned GMR, in a reaching task constrained by an obstacle.

In [25], the PoWER algorithm is introduced, based on Expectation-Maximization. It is applied to learning the "ball in a cup" game, using a DMP as a parameterized policy. A similar algorithm was used in [5] in a pancake flipping task using a different parameterized policy consisting of a time-dependent mixture of proportional-derivative systems.

PI^2 is a policy search method derived from the first principles of stochastic optimal control [14]. The update rule takes the form of a probability-weighted average and shares some similarities with the PoWER algorithm. It is used in [16] to learn a variety of robot motion tasks using a DMP policy. The reference trajectory and the stiffness profile are learned simultaneously for an impedance control architecture. In [9], and adaptation of PI^2 called PI^2-01 is used to refine a parameterized policy based on a mixture of functions that are affine with respect to the state variables. The mixing factor is time dependent. One particularity of this approach is that the parameterized policy produces directly low level controls (i.e. the motor torques to be applied) instead of a reference trajectory. The policy can be viewed as providing both feed-forward and feedback terms together.

While these studies have all demonstrated learning of complex tasks, they all use motion representations that depend implicitly on time and hence suffer from the problems described in Section 2.1.

A wider panel of parameterized policies, including time-invariant ones, can be found in other examples that are related to motion learning but that take the approach of learning a reactive behavior rather than a trajectory. Examples include learning a tracking controller for a helicopter and a balancing robot [26], learning a reactive controller for peg-in hole insertion [27], and learning the non-episodic task of walking [28]. Those examples have the limitation that the non-linear features used for the policy or the value function need to be defined a priory. In contrast, we leverage on LfD to use demonstrations for building the relevant features.

2.3 Learning varying stiffness control

Recent years have seen a revitalized interest in the role of impedance control [29] when designing robot controllers for uncertain environments including physical interaction. A lot can be learned from biological systems, where numerous studies report that humans change the stiffness of their arms not only between different tasks but also during the execution of a task [30–32]. Several related robot controllers have been proposed. The principles observed in human impedance adaptation were transfered to an adaptive impedance controller in [33]. Task specific impedance principles captured via a learned cost-function were transferred from humans to robots in [34], and direct mirroring of human stiffness was proposed for tele-operation applica-

tions in [35]. In the field of LfD, it has been proposed to derive stiffness variations via kinematic variability in demonstrated data [36]. Recent works have taken force measurements into account for estimating stiffness using weighted least squares [37] and least squares with platform specific priors on the stiffness parameters [38]. Another approach developed dedicated Human-Robot interfaces for the purpose of enabling stiffness variations to be easily taught to a robot [17]. Probabilistic motion representations [39,40] can be used to derive varying feedback gains in a principled manner [41,42,48]. Optimal control has been used successfully to exploit passive dynamics of robotic devices with intrinsically variable stiffness [43,44].

In all of the works cited above, learning stiffness is a one-shot process, often using a set of demonstrations to derive a varying stiffness profile. However the robot is not given the possibility to improve its gain schedule over time. As has been demonstrated in all these works, it is possible to derive stiffness that increase task performance using any of these techniques. However, except the works using optimal control, none of these methods ensure that the best possible gain schedule is learned. Furthermore, if a change in the environment necessitates a correction in the gain schedule, additional demonstrations have to be provided.

To deal with this problem, we use reinforcement learning to allow the robot to improve over time, possibly in ways that were difficult or impossible for the user to demonstrate [17]. The price we pay for this added capability is the need for a well-designed cost function describing the goal of the task. We take the approach pioneered in [16] based on learning varying stiffness and reference motion simultaneously via reinforcement learning. We differ in that our representation of learned reference motion and stiffness is state-dependent as opposed the time-dependent formulation used in [16].

3 Integrating LfD and RL using time-invariant DS based policies

The first part of this section explains how an initial parameterized policy can be extracted from a set of demonstrations. This initial policy provides both a set of initial parameters and a set of hyper-parameters that determine the non-linear features used by the policy. This is achieved by means of a Gaussian Mixture Regression (GMR), which yields a policy of the form of a non-linear time-invariant dynamical system.

The second part of this section explains how the resulting GMR policy can be refined using a version of PI^2 adapted to the case of time-invariant policies.

The third part presents simulation to illustrate the functioning and the performance of the proposed method.

3.1 Extracting an initial policy from demonstration

GMR are multivariate regressions based on conditioned Gaussian Mixture Models (GMM) [21]. The GMM is trained to approximate the joint probability distribution $p(\boldsymbol{x}, \dot{\boldsymbol{x}})$ of the position $\boldsymbol{x}$ and velocity $\dot{\boldsymbol{x}}$ from the demonstration dataset using an Expectation-Maximization algorithm. GMR can then be used to compute an estimation of the reference velocity for any given position, yielding the following DS:

$$\begin{aligned}\dot{\boldsymbol{x}} &= GMR(\boldsymbol{x}) = E\{p_{GMM}(\dot{\boldsymbol{x}}|\boldsymbol{x})\} \\ &= \sum_{k=1}^{N_G} h_{x_t}^k(\boldsymbol{\mu}_{\dot{x}}^k + \boldsymbol{\Sigma}_{\dot{x}x}^k(\boldsymbol{\Sigma}_x^k)^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_x^k)) \\ &= \sum_{k=1}^{N_G} h_{x_t}^k(\boldsymbol{A}^k\boldsymbol{x} + \boldsymbol{b}^k),\end{aligned} \tag{1}$$

where N_G is the given number of Gaussian functions in the mixture, and

$$\begin{aligned}h_{x_t}^k &= \frac{\pi^k N(\boldsymbol{x}; \boldsymbol{\mu}_x^k, \boldsymbol{\Sigma}_x^k)}{\sum_{i=1}^{N_G}\pi^i N(\boldsymbol{x}; \boldsymbol{\mu}_x^i, \boldsymbol{\Sigma}_x^i)}, \\ \boldsymbol{A}^k &= \boldsymbol{\Sigma}_{\dot{x}x}^k(\boldsymbol{\Sigma}_x^k)^{-1}, \\ \boldsymbol{b}^k &= \boldsymbol{\mu}_{\dot{x}}^k - A^k\boldsymbol{\mu}_x^k,\end{aligned} \tag{2}$$

and $\boldsymbol{\mu}_x^k, \boldsymbol{\mu}_{\dot{x}}^k, \boldsymbol{\Sigma}_x^k, \boldsymbol{\Sigma}_{\dot{x}x}^k$ are parts of the vector of means and of the covariance matrices of the Gaussians forming the GMM:

$$\boldsymbol{\mu}^k = \begin{bmatrix}\boldsymbol{\mu}_x^k \\ \boldsymbol{\mu}_{\dot{x}}^k\end{bmatrix}, \qquad \boldsymbol{\Sigma}^k = \begin{bmatrix}\boldsymbol{\Sigma}_x^k & \boldsymbol{\Sigma}_{\dot{x}x}^k \\ \boldsymbol{\Sigma}_{\dot{x}x}^k & \boldsymbol{\Sigma}_{\dot{x}}^k\end{bmatrix}. \tag{3}$$

A reference trajectory can then be generated by integration:

$$\boldsymbol{x}_{t_{i+1}} = \boldsymbol{x}_{t_i} + \sum_{k=1}^{N_G} h_{x_{t_i}}^k(\boldsymbol{A}^k\boldsymbol{x}_{t_i} + \boldsymbol{b}^k)\Delta t. \tag{4}$$

This procedure is illustrated by two examples in Figure 1. Two different motions are learned from sets of three demonstrations each, using four Gaussian basis functions ($N_G = 4$).

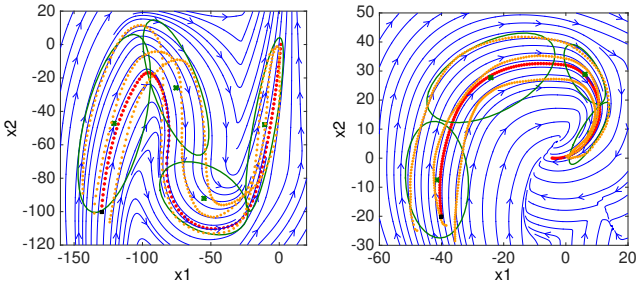


Fig. 1 The demonstration trajectories (orange) are used to learn a GMR policy (the streamlines represent the velocity for any given position) using 4 Gaussians (the ellipses show the 1 standard deviation state space projection of the associated Gauss function), which produces a reference trajectory (red) for a starting position in the demonstration area (black square), and a given number of time steps.

3.2 Path integral policy improvement method for time-invariant DS based policy

PI^2 [14] is a policy search RL algorithm derived from stochastic optimal control. It assumes a dynamical system controlled by a parameterized policy which is linear in the learned parameter vector $\boldsymbol{\theta}$:

$$\dot{\boldsymbol{x}}_{\boldsymbol{t}} = \boldsymbol{f}(\boldsymbol{x}_{\boldsymbol{t}}) + \boldsymbol{G}_{\boldsymbol{x}_{\boldsymbol{t}}}(\boldsymbol{\theta} + \boldsymbol{\epsilon}_t), \tag{5}$$

where $\boldsymbol{x}_t$ is the state of the system at time t, $\boldsymbol{f}(\boldsymbol{x}_{\boldsymbol{t}})$ is the transition function, $\boldsymbol{G}_{\boldsymbol{x}_{\boldsymbol{t}}}$ the control matrix and $\boldsymbol{\epsilon}_t$ is a Gaussian exploration noise vector with zero mean and covariance matrix $\boldsymbol{\Sigma}_\epsilon$.

The cost of a trajectory $\boldsymbol{\tau}^{\boldsymbol{\theta}}_{\boldsymbol{x}_{t_i}} = [\boldsymbol{x}_{t_i}\boldsymbol{x}_{t_{i+1}}\dots\boldsymbol{x}_{t_N}]_{\boldsymbol{\theta}}$ starting at time t_i in state $\boldsymbol{x}_{t_i}$ and resulting from the policy parameterized by $\boldsymbol{\theta}$ is given by:

$$C(\boldsymbol{\tau}^{\boldsymbol{\theta}}_{\boldsymbol{x}_{t_i}}) = \Phi(\boldsymbol{x}_{t_N}) + \int_{t_i}^{t_N} (q(\boldsymbol{x}_{t_i}) + \frac{1}{2}\boldsymbol{u}_t^T \boldsymbol{R}\boldsymbol{u}_t), \tag{6}$$

with $\Phi(\boldsymbol{x}_{t_N})$ the terminal cost, $q(\boldsymbol{x}_{t_i})$ the immediate cost, $\boldsymbol{u}_t = \boldsymbol{G}_{\boldsymbol{x}_{\boldsymbol{t}}}(\boldsymbol{\theta} + \boldsymbol{\epsilon}_t)$ the control given by the policy and $\boldsymbol{R}$ the control cost matrix.

The objective is to find the parameter vector $\boldsymbol{\theta}$ that minimizes the value function:

$$V(\boldsymbol{x}_{t_i}, t_i) = \mathrm{E}[C(\tau^{\boldsymbol{\theta}}_{\boldsymbol{x}_{t_i}})], \tag{7}$$

where the expectation is taken over all the stochastic trajectories $\tau^{\boldsymbol{\theta}}_{\boldsymbol{x}_{t_i}}$ starting from $\boldsymbol{x}_{t_i}$ and subject to the policy parameterized by $\boldsymbol{\theta}$.

An iterative scheme is used to improve $\boldsymbol{\theta}$. At each iteration a batch of trajectories (called rollouts), which differ due to the stochasticity of $\boldsymbol{\epsilon}_t$, are sampled and used to compute an update $\delta\boldsymbol{\theta}$ for the parameter vector, following a path integrals approach.

PI^2 is generally used to learn parameterized policies that generate a reference trajectory that a system will track. It is shown that the dynamics of the actual system tracking the trajectory is not relevant to the learning algorithm. The state vector $\boldsymbol{x}$ thus only contains the states describing the reference trajectory (in this paper $\boldsymbol{x} \in \mathbb{R}^2$, except for the hardware experiment).

The major difference between our approach and [14] is that we use the GMR parameterized policy presented in subsection 3.1 instead of the DMP parameterized policy used in the original algorithm. Our policy is thus state dependent and time-invariant, which is a fundamental change.

The GMR policy in Eq. (1) can be rewritten in the form of Eq. (5), and we identify $\boldsymbol{G}_{x_t}$ and $\boldsymbol{\theta}$ to be the following (shown here for the 2-dimensional case):

$$\boldsymbol{G}_{x_t} = \left[\boldsymbol{G}^1_{x_t}, \boldsymbol{G}^2_{x_{t_i}}, \dots, \boldsymbol{G}^{N_G}_{x_t}\right], \tag{8}$$

$$\boldsymbol{G}^k_{x_t} = h^k_{x_t} \begin{bmatrix} x_{1,t}, x_{2,t}, & 0, & 0, & 1, 0 \\ 0, & 0, & x_{1,t}, x_{2,t}, & 0, 1 \end{bmatrix}, \tag{9}$$

$$\boldsymbol{\theta} = \left[\boldsymbol{\theta}^1, \boldsymbol{\theta}^2, \dots, \boldsymbol{\theta}^{N_G}\right]^T, \tag{10}$$

$$\boldsymbol{\theta}^k = \left[A^k_{1,1}, A^k_{1,2}, A^k_{2,1}, A^k_{2,2}, b^k_1, b^k_2\right]. \tag{11}$$

At this point we see that the parameters that are learned (i.e the components of $\boldsymbol{\theta}$) are the parameters of the matrices $\boldsymbol{A}^k$ and vectors $\boldsymbol{b}^k$ as defined in Eq. (2). As the terms $\boldsymbol{\mu}^k_x$ and $\boldsymbol{\Sigma}^k_x$ are also in $h^k_{x_t}$, they cannot be part of the learned parameters because of their non-linear effect in the DS (i.e. it would not respect the form of Eq. 5). Thus, what is actually learned is the mean of the velocity $\boldsymbol{\mu}^k_{\dot{x}}$ and the covariance between position and velocity $\boldsymbol{\Sigma}^k_{\dot{x}x}$. The input part of the centers $\boldsymbol{\mu}^k_x$ and the shapes of the Gaussians $\boldsymbol{\Sigma}^k_x$ remain fixed (initialized by LfD as described in Section 3.1).

The control matrix $\boldsymbol{G}_{x_t}$ contains the non-linear features of the policy. The features consist of the Gaussian basis functions multiplied by the input variables. The Gaussian basis functions are positioned in the input space during policy initialization through LfD (refer to Section 3.1).

It should be noted that for the parameterization given above (Equations 8 to 11) $\boldsymbol{f}(x_t)$ is set to zero in Eq. (5). Another possible parametrization is

$$\boldsymbol{f}(\boldsymbol{x}_t) = \sum_{k=1}^{N_G} h^k_{x_t}(\boldsymbol{A^k}^0 \boldsymbol{x}_t + \boldsymbol{b^k}^0), \tag{12}$$

$$\boldsymbol{\theta}^k = \left[{A^k_{1,1}}^*, {A^k_{1,2}}^*, {A^k_{2,1}}^*, {A^k_{2,2}}^*, b^{k*}_1, b^{k*}_2\right], \tag{13}$$

where ${A^k_{i,j}}^0$ and b^{k0}_i are the original parameters of the GMR learned from the demonstration dataset. ${A^k_{i,j}}^* = A^k_{i,j} - {A^k_{i,j}}^0$ and $b^{k*}_i = b^k_i - b^{k0}_i$ denote the difference between the original parameters and the current parameters. Under this parametrization, the original GMR is

seen as the system's passive dynamics, and the control is the modification to the original GMR. $\boldsymbol{G}_{x_t}$ remains unchanged.

With the parameterization in Eq. (11), the policy search algorithm will try to keep the norm of the parameters small. In contrast, with the parameterization given by Eq. (13), the control cost will tend to keep the parameters relatively close to their initial values learned from demonstration. The most appropriate choice may depend on the task, but Eq. (13) appears as the most natural choice for most cases and is used in the remainder of this paper.

The derivation of the update rule for PI^2 with the GMR parameterized policy follows closely the original derivation of the update rule for PI^2 with the DMP parameterized policy as described in [14] up to the calculation of the optimal control (denoted $\boldsymbol{u}^*_{t_i}$ in [14]).

After obtaining the optimal control, a parameter update $\delta\boldsymbol{\theta}_{t_i}$ can be computed for each time step. To get a single update vector $\delta\boldsymbol{\theta}$, a time averaging method can then be used as proposed in the original PI^2 algorithm (see [14]). Alternatively, one can consider only the first parameter update ($\boldsymbol{\delta\theta} = \boldsymbol{\delta\theta}_{t_1}$), as proposed in the PI$^{\mathrm{BB}}$ algorithm (see [15]). We chose the latter as it avoids projection of the exploration noise on the range space of the control matrix, which is not suitable in the GMR representation (as shown later in Figure 3). The complete procedure is described in algorithm 1.

Algorithm 1 PI^2 for GMR parameterized policy

Given
- a set of demonstration trajectories *Demos*
- the covariance $\boldsymbol{\Sigma}_\epsilon$ of the exploration noise $\boldsymbol{\epsilon}$
- a cost function composed of:
  - a terminal cost term $\Phi(\boldsymbol{x}_{t_N})$
  - an immediate cost term $q(\boldsymbol{x}_{t_i})$
  - a quadratic control cost specified by the control matrix $\boldsymbol{R}$
- a transition function $\boldsymbol{f}(x)$ ($= \mathbf{0}$ or given by equ. 12)

procedure INITIALIZE(*Demos*)
  GMR$\leftarrow$ E-M(*Demos*)
  for k $\leftarrow 1, N_G$ **do**
    $\boldsymbol{\theta}^k = [{A^k_{1,1}}^* \quad {A^k_{1,2}}^* \quad {A^k_{2,1}}^* \quad {A^k_{2,2}}^* \quad b^{k*}_1 \quad b^{k*}_2]$
  end for
  $\boldsymbol{\theta} = [\boldsymbol{\theta}^1 \quad \boldsymbol{\theta}^2 \quad \dots \quad \boldsymbol{\theta}^{N_G}]$
end procedure
procedure POLICY SEARCH($\boldsymbol{\theta}$, Cost function)
  while Cost has not converged **do**
    $\alpha \leftarrow \max\left(\frac{1}{n_{iter}}, 0.1\right)$
    for $r \leftarrow 1, N_r$ **do**
      $\boldsymbol{\theta}^r \leftarrow \boldsymbol{\theta} + \boldsymbol{\epsilon}^r, \quad \boldsymbol{\epsilon}^r \in \mathcal{N}(0, \alpha\boldsymbol{\Sigma}_\epsilon)$
      $\boldsymbol{\tau}^r \leftarrow [\boldsymbol{x}_{t_1} \quad \dots \quad \boldsymbol{x}_{t_N}]_{\boldsymbol{\theta}^r}$ ▷ Sample trajectory
      $\boldsymbol{M_{t_i,r}} \leftarrow \boldsymbol{R}^{-1}\boldsymbol{G}^T_{t_i,r}(\boldsymbol{G}_{t_i,r}\boldsymbol{R}^{-1}\boldsymbol{G}^T_{x_{t_i}})^{-1}\boldsymbol{G}_{t_i,r}$
      $\Upsilon_{t_i,r} \leftarrow (\boldsymbol{\theta} + \boldsymbol{M}_{t_i,r}\boldsymbol{\epsilon}_{t_j,r})^T\boldsymbol{R}(\boldsymbol{\theta} + \boldsymbol{M}_{t_i,r}\boldsymbol{\epsilon}_r)$
      $S(\boldsymbol{\tau}^r) \leftarrow \Phi_{x_{t_N},r} + \sum_{j=0}^{N_t} q_{xt_j,r} + \frac{1}{2}\sum_{j=0}^{N_t}\Upsilon_{t_i,r}$
    end for
    $\lambda \leftarrow \frac{\max_r S(\boldsymbol{\tau}^r) - \min_r S(\boldsymbol{\tau}^r)}{10}$
    for $r \leftarrow 1, N_r$ **do**
      $P(\boldsymbol{\tau}^r) \leftarrow \frac{e^{-\frac{1}{\lambda}S(\boldsymbol{\tau}^r)}}{\sum_{r=1}^{N_r} e^{-\frac{1}{\lambda}S(\boldsymbol{\tau}^r)}}$
    end for
    $\delta\boldsymbol{\theta} \leftarrow \sum_{r=1}^{N_r} P(\boldsymbol{\tau}^r)\boldsymbol{\epsilon}^r$ ▷ Compute update
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{\delta\theta}$ ▷ Update parameter vector
  end while
end procedure

3.3 Simulation experiments

To illustrate the performance of the algorithm, we present two simulation experiments.

In the first experiment we apply the algorithm to the task of reaching a goal $\boldsymbol{x}^{goal}$ from an initial state while traveling through a set of via-points $\boldsymbol{x}^{via}_l, l = 1 \dots N_{VP}$. The via points are distributed along the general motions demonstrated earlier, with the goal at the end. The policy is initialized with the GMR previously learned from demonstration. This can be seen as refining an imperfectly demonstrated motion for a specific performance criterion. Details of this simulation experiment can be found in Appendix A.1.

Figure 2 shows the final policies and resulting trajectories that are obtained after applying the RL procedure. It can be seen that the policies are adapted so that the trajectories pass through the via-points. The learning curves show how the different component of the cost evolve along the iterations. There is a fast initial decrease in the via-point cost, while the acceleration cost increases slowly. The final cost results from a trade-off of those two components. As for the control cost, it only plays a regularization role.

We also show that, as discussed previously in part 3.2, the PI$^{\mathrm{BB}}$ way of computing the update term is more suitable to our parameterized policy than the PI^2 way, yielding significantly lower final costs as seen in Figure 3.

In the second simulation experiment, we illustrate one advantage we gain in using time-invariant dynamical systems as parameterized policies : the ability to learn different behaviors for different parts of the state space.

This time, the policy is evaluated for two different initial states of the system. At each roll-out, there are thus two different trajectories. The two trajectories are evaluated with the same cost function, and the costs are added to obtain the cost of the roll-out. Details of this simulation experiment can be found in Appendix A.2.

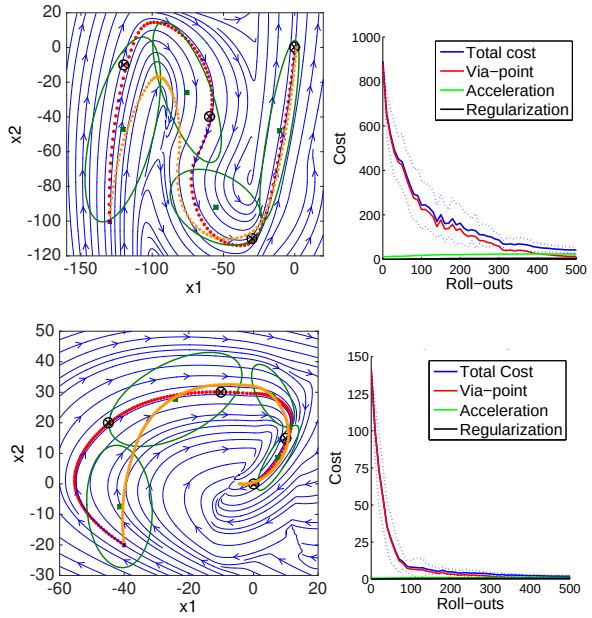


Fig. 2 Left: The final GMR policies for the N-shape (top) and P-shape (bottom) after the RL procedure, which produce reference trajectories (red) for a starting position (black square). The via-points and goals are depicted by black crossed circles. The initial trajectory learned from demonstration are shown for comparison (orange). Right: The learning curves showing the cost mean along the roll-outs (one iteration is ten roll-outs).

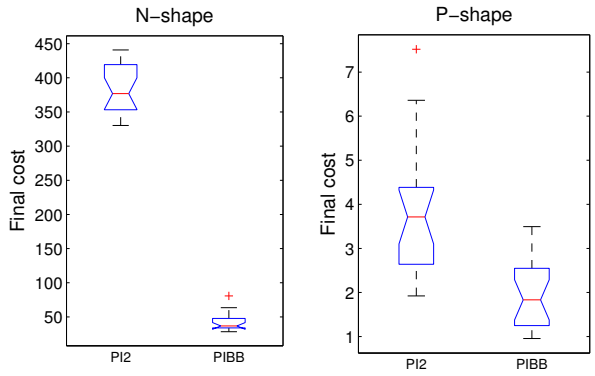


Fig. 3 Boxplots comparing the cost at the final iteration over 20 runs when using the the PI^2 or PI^{BB} ways of computing the update term in the N-shape and P-shape simulation scenarios. The PI^{BB} way is more suitable to the time-invariant DS parameterization of the policy.

Initially, the system behaves similarly in the left and right parts of the state space because it has received similar (but mirrored) demonstrations in both parts to build the initial policy. Through reinforcement learning, the policy is updated and specific behaviors appear for the left and right parts of the state space to fit the specific via-points layout, as seen in Figure 4. The time-invariant DS parameterization of the policy is key to enable this behavior.

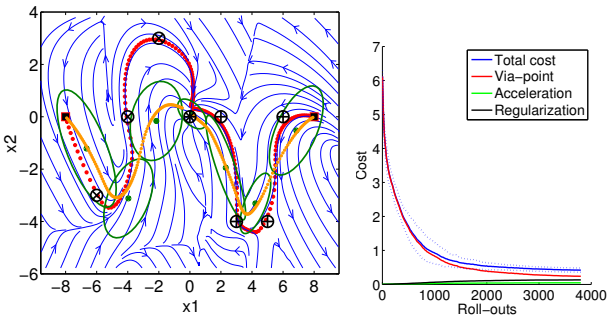


Fig. 4 Simulation experiment with two different starting states (back squares). The goal state is at the center (0,0) and there are two sets of via-points (black circles with x or + crosses) in the right and left parts of the state space that have a different arrangement. Starting form an initial policy that yields two similar initial trajectories (orange), the RL procedure refines it and adapts the behavior so that the trajectories pass through the via-points. The final policy (blue stream lines) displays very different trajectories in the right and left parts of the state space, illustrating the advantage of having a policy parameterization that enables state dependent behavior.

4 Shaping the exploration noise

Exploration is a critical factor of RL, which can have just as much impact on the performance of the algorithm as the update procedure. Different tasks may require different exploration strategies and sometimes task knowledge can be used to focus the exploration on promising parts of the policy space. In this section we describe two different exploration strategies adapted to our parameterization.

4.1 Homogeneous exploration in task space

The tuning of the exploration noise is critical to obtain good performance in RL. When too low, the exploration will not be sufficient and the improvement will be very slow and may stop prematurely in a local minimum. When too high, there is a high chance that the sampled policies will perform poorly because they are too far from the region of interest. In addition, the relative exploration noise applied to each different parameter can also be important. In order to define a suitable exploration noise for our parameterized policy, we analyze the effect of the exploration noise ϵ in the task space (i.e. the perturbation on the reference velocity).

For simplicity we consider a region where only a single Gaussian basis function k is active. In these conditions the noise $\epsilon_{\dot{v}_j^{ref}}$ on the jth component of the reference velocity vector can be written as

$$\epsilon_{\dot{v}_j^{ref}} = \epsilon_{b_j^k} + \sum_{d=1}^{N_{in}} x_d^{ref}\, \epsilon_{A_{j,d}^k} \,. \tag{14}$$

As the noise $\epsilon_{b_j^k}$ and $\epsilon_{A_{j,d}^k}$ applied to the parameters is drawn form Gaussian distributions with zero mean, each $\epsilon_{v_j^{ref}}$ has also a Gaussian distribution with zero mean and variance $\sigma^2_{v_j^{ref}}$ calculated as

$$\sigma^2_{v_j^{ref}} = \sigma^2_{b_j^k} + \sum_{d=1}^{N_{in}} {x_d^{ref}}^2 \, \sigma^2_{A_{j,d}^k} \, . \tag{15}$$

Thus the velocity noise vector is a random Gaussian vector. If we do not have prior knowledge of the task, we generally would like the variance of the velocity noise vector to be isotropic so that all dimensions are explored equally, and also state invariant so that all parts of the trajectory can be equally perturbed. Isotropy can be achieve by choosing $\sigma^2_{b_j^k} = \sigma^2_{b^k}$ and $\sigma^2_{A_{j,d}^k} = \sigma^2_{A_d^k} \quad \forall j = 1 : N_{out}$.

A way of partially achieving state invariance is to tune the variance of the noise for each Gaussian basis function such that it compensates for the state dependency. This can be done by choosing $\sigma^2_{b^k} = \sigma^2_{b^0}$ and $\sigma^2_{A_d^k} = (\sigma_{A_d^0}/\mu_d^k)^2$, where μ_d^k is the mean of the kth Gaussian basis function in the dth input dimension. This way the state dependency of the variance of the velocity noise is exactly canceled at the center of the Gaussian basis functions and partially compensated elsewhere in the vicinity of the basis functions.

The mean norm of the velocity noise vector is a relevant parameter for specifying the exploration noise. With our choices so far and for the 2-dimensional case[1], the norm of the velocity noise vector follows a Rayleigh distribution and its mean is $\mu_v = \tilde{\sigma}\sqrt{\frac{\pi}{2}}$ with $\tilde{\sigma} = \sqrt{\sigma^2_{b^0} + \sum_{d=1}^{N_{in}} \sigma^2_{A_d^0}}$. Thus, the mean norm of the velocity noise vector can be set to a desired value μ_v^{des} by choosing $\sigma^2_{A_d^0}$ and $\sigma^2_{b^0}$. We choose to share the responsibility of producing the noise vector equally among all the terms composing μ_v^{des}, so $\sigma^2_{b^0} = \sigma^2_{A_d^0} = \frac{\tilde{\sigma}^2}{3} = \frac{2\mu_v^2}{3\pi}$.

With all the choices explained in this sub-section, we obtain the following rules for tuning the noise for the 2-dimensional case:

$$\sigma^2_{A_{1,1}^k} = \sigma^2_{A_{1,2}^k} = \frac{2{\mu_v^{des}}^2}{3\pi {\mu_1^k}^2} \, , \tag{16}$$

$$\sigma^2_{A_{2,1}^k} = \sigma^2_{A_{2,2}^k} = \frac{2{\mu_v^{des}}^2}{3\pi {\mu_2^k}^2} \, , \tag{17}$$

$$\sigma^2_{b_1^k} = \sigma^2_{b_2^k} = \frac{2{\mu_v^{des}}^2}{3\pi} \, , \tag{18}$$

[1] for the 3-dimensional case, the norm follows a Maxwell-Boltzmann distribution with mean $\mu_v = 2\tilde{\sigma}\sqrt{\frac{2}{\pi}}$

where μ_v^{des} is the desired mean norm of the velocity noise vector and μ_i^k is the mean of the kth Gaussian basis function in the ith input dimension.

Although those rules are derived for the simplified case where only one Gaussian basis function is active, they also give reasonable results in locations in the state space where several basis functions affect the control.

Figure 5 compares the mean value of the velocity noise vector for the different parts of the state space when using a uniform variance for the noise on each parameter and when using the method described in this subsection. In the case of uniform noise on the parameters, we can see how the velocity noise grows with the distance to the origin (0;0), with some additional patterns caused by the mixture of the Gaussian basis functions. This growth is undesirable. When using the proposed tuning method, the velocity noise is controlled in the vicinity of each Gaussian basis function, and remains within 50% of the desired value in the area visited by the trajectory.

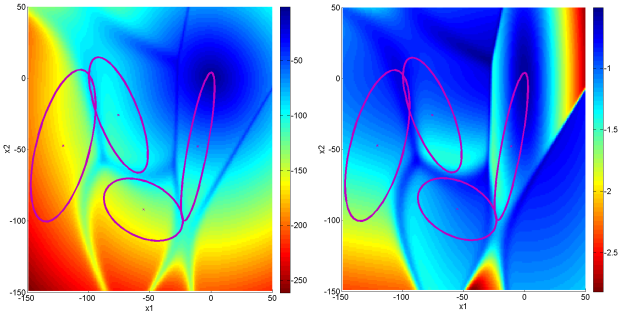


Fig. 5 Maps of the mean norm of the velocity noise vector for the N shape scenario (the 1 standard deviation state space projection of the Gaussian basis function is shown in magenta). Left: using uniform variance of 1 for all the parameters. Right: using the tuning method with $\mu_v^{des} = 1$

4.2 Stable exploration

Using a DS encoded by a GMR as a reference trajectory allows to represent a large variety of trajectories. However, without parameter constraints the result is generally an unstable model with diverging trajectories. This can impact the learning as many roll-outs can leave the area of interest and hence provide little or no relevant information. In [7], a set of constraints that ensure a stable model were presented. Here we give a brief review of these constraints and propose a sampling method that ensures that a stable model remains stable during RL.

Recalling that a GMR encodes a dynamical system as a mixture of linear systems of the form:

$$\dot{\boldsymbol{x}} = \sum_{k=1}^{N_G} h_{x_t}^k (\boldsymbol{A}^k \boldsymbol{x} + \boldsymbol{b}^k)\,,$$

the SEDS conditions [7] are expressed as follows:

1. $\boldsymbol{b}^k = -\boldsymbol{A}^k \boldsymbol{x}^{goal} \quad \forall k = 1 : N_G$
2. $\frac{\boldsymbol{A}^k + \boldsymbol{A}^{k\,T}}{2} \prec 0 \quad \forall k = 1 : N_G$

where $\boldsymbol{x}^{goal}$ is the goal state and N_G is the number of Gaussian basis functions used for the GMR.

In order to make sure that each iteration of the RL algorithm preserves the SEDS conditions, we will shape the exploration noise applied to the $\boldsymbol{A}^k$ matrix ($\boldsymbol{\epsilon}_A^k$) and $\boldsymbol{b}^k$ vector ($\boldsymbol{\epsilon}_b^k$) so that the SEDS conditions are preserved for each roll-out r:

1. $\boldsymbol{b}^{k,r} = -\boldsymbol{A}^{k,r} \boldsymbol{x}^* \quad \forall k = 1 : N_G, r = 1 : N_r$
2. $\frac{\boldsymbol{A}^{k,r} + \boldsymbol{A}^{k,r\,T}}{2} \prec 0 \quad \forall k = 1 : N_G, r = 1 : N_r$

with

$$\boldsymbol{b}^{k,r} = \boldsymbol{b}^k + \boldsymbol{\epsilon}_b^{k,r}\,, \tag{19}$$

$$\boldsymbol{A}^{k,r} = \boldsymbol{A}^k + \boldsymbol{\epsilon}_A^{k,r}\,, \tag{20}$$

where the vector $\boldsymbol{\epsilon}_b^{k,r}$ and the matrix $\boldsymbol{\epsilon}_A^{k,r}$ are the elements of the exploration noise vector $\boldsymbol{\epsilon}^r$ that correspond to the parameters forming $\boldsymbol{A}^k$ and $\boldsymbol{b}^k$.

Note that $\frac{\boldsymbol{A}^k + \boldsymbol{A}^{k\,T}}{2}$ is the symmetric part of the matrix $\boldsymbol{A}^k$, that can be written in terms of its symmetric part and antisymmetric part as:

$$\boldsymbol{A}^k = \frac{\boldsymbol{A}^k + \boldsymbol{A}^{k\,T}}{2} + \frac{\boldsymbol{A}^k - \boldsymbol{A}^{k\,T}}{2}\,. \tag{21}$$

The second SEDS condition only constraints the symmetric part of $\boldsymbol{\epsilon}_A^{k,r}$, and therefore the antisymmetric part, which we call $\boldsymbol{\epsilon}_{anti}^{k,r}$, can be built from a random Gaussian noise vector with desired variance.

The symmetric part of $\boldsymbol{\epsilon}_A^{k,r}$, which we call $\boldsymbol{\epsilon}_{sym}^{k,r}$, must be built such that the symmetric part of $\boldsymbol{A}^{k,r}$ remains negative definite. Constraining $\boldsymbol{\epsilon}_{sym}^{k,r}$ to be negative definite would be sufficient but is overly restrictive, as it would only allow $\boldsymbol{A}^k$ to grow in magnitude.

It can be shown that for a negative definite matrix $\boldsymbol{N}$ and a symmetric (not necessarily definite) matrix $\boldsymbol{S}$, one can always find small enough $\eta > 0$ such that $\boldsymbol{N} + \eta \boldsymbol{S}$ remains negative definite. We use this fact to build $\boldsymbol{\epsilon}_{sym}^{k,r}$ by generating a symmetric matrix $\boldsymbol{S}^{k,r}$ from a Gaussian noise vector with desired variance, and checking the negative definiteness of $\frac{\boldsymbol{A}^k + \boldsymbol{A}^{k\,T}}{2} + \eta \boldsymbol{S}^{k,r}$ for η decreasing from 1 to 0 until check is passed. The exploration noise applied to $\boldsymbol{A}^k$ is then:

$$\boldsymbol{\epsilon}_A^{k,r} = \boldsymbol{\epsilon}_{sym}^{k,r} + \boldsymbol{\epsilon}_{anti}^{k,r}\,. \tag{22}$$

The exploration noise applied to $\boldsymbol{b}^k$ is imposed by the first SEDS condition:

$$\boldsymbol{\epsilon}_b^{k,r} = -\boldsymbol{\epsilon}_A^{k,r} \boldsymbol{x}^*\,. \tag{23}$$

Having built the exploration noise such that the policy of each roll-out satisfies the SEDS conditions, we now show that the update process (lines 9 to 26 of algorithm 1) preserves those conditions. The update vector $\delta\boldsymbol{\theta}$ is a weighted sum of the exploration noise of the different roll-outs. If we remap this vector to $\boldsymbol{A}^k$ and $\boldsymbol{b}^k$ (reversing lines 4 to 7 of algorithm 1), and considering lines 23 and 25 of algorithm 1, we can write:

$$\boldsymbol{b}_{new}^k = \boldsymbol{b}^k + \delta\boldsymbol{b}^k\,, \tag{24}$$

$$\delta\boldsymbol{b}^k = \sum_{r=1}^{N_r} P(r) \boldsymbol{\epsilon}_b^{k,r}\,, \tag{25}$$

$$\boldsymbol{A}_{new}^k = \boldsymbol{A}^k + \delta\boldsymbol{A}^k\,, \tag{26}$$

$$\delta\boldsymbol{A}^k = \sum_{r=1}^{N_r} P(r) \boldsymbol{\epsilon}_A^{k,r}\,, \tag{27}$$

and we need $\boldsymbol{b}_{new}^k$ and $\boldsymbol{A}_{new}^k$ to satisfy the SEDS conditions.

It can be shown that this is the case:

$$\boldsymbol{A}_{new}^k = \boldsymbol{A}^k + \sum_{r=1}^{N_r} P(r) \boldsymbol{\epsilon}_A^{k,r} = \sum_{r=1}^{N_r} P(r)(\boldsymbol{A}^k + \boldsymbol{\epsilon}_A^{k,r}) \prec 0\,. \tag{28}$$

The second equality of Eq. 28 holds because $P(r)$ is normalized (i.e. $\sum_{r=1}^{N_r} P(r) = 1$), and the negative definiteness comes from the fact that $\boldsymbol{A}^k + \boldsymbol{\epsilon}_A^{k,r}$ is negative definite by construction and that a convex combination of negative definite matrices is negative definite as well.

4.3 Simulation results for stable exploration

We illustrate the functioning of our algorithm in combination with the sampling method proposed in subsection 4.2 with a 2-dimensional constrained reaching task. In this task, the agent starts in a given position and has to reach the goal within a given time (15sec, $\Delta t = 0.1$, $N_t = 150$) and with the shortest possible path. The goal is in a box and the agent needs to take a detour to reach it, as illustrated in Figure 6. If the agent collides with the box, its motion is stopped until the end of the task.

A few sub-optimal demonstration trajectories are given that span an area containing the starting position. An initial SEDS policy is learned from those demonstrations using the SEDS algorithm. This initial policy is then used as a starting point for the reinforcement

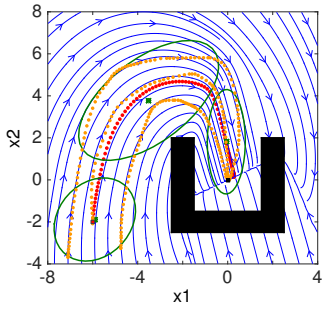


Fig. 6 Constrained reaching scenario with a goal (black square) in a box (in black). The demonstration trajectories (in dashed/orange) are used to learn a SEDS policy (the streamlines represent the velocity for any given position) using 3 Gaussian basis functions (the ellipses show the 1 standard deviation state space projection of the associated Gauss function), which produces a reference trajectory (in red/bold) for a starting position (red/bold square).

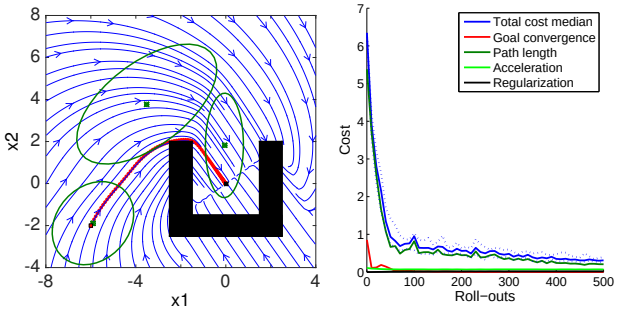


Fig. 7 Left: final policy after reinforcement learning and resulting reference trajectory. Right: Learning curve along the iterations of the RL procedure, showing the median value for the different components of the cost.

learning algorithm. Details of this simulation experiment can be found in Appendix B.

The policy and trajectory resulting form the reinforcement learning and the learning curve along the iterations is shown in Figure 7. It can be seen that the reinforcement learning procedure is capable of reducing the length of the trajectory significantly after 100 roll-outs already (which means 10 updates). The cost continues decreasing slowly until about 400 roll-outs.

Figure 8 shows that the presented noise shaping method increases the efficiency of the learning for this task. The learning curve with and without the SEDS constraints are compared and it appears clearly that the cost decreases faster and lower with the SEDS constraints. Our interpretation is that enforcing the stability of the trajectory at every roll-out is particularly helpful in this task where the shape of the trajectory is restricted. Without the SEDS constraints, to many roll-outs are lost with poor performance because of collisions with the box.

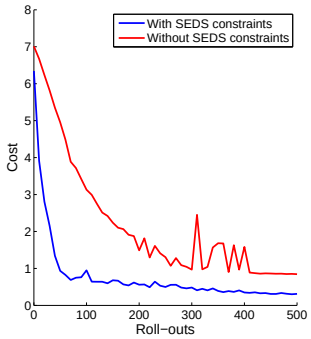


Fig. 8 The learning curves (medium values on 20 runs) for the constrained reaching scenario with (blue plain) and without (red dashed) the SEDS constraints on the exploration noise, which insure stability of the trajectory at each roll-out.

4.4 Summary on exploration noise shaping

We introduced two different exploration noise shaping methods tailored to our policy parameterization. The aim of these two methods is to obtain two exploration strategies that are suitable for different situations.

The first method promotes the isotropy and state independence of the velocity noise vector, which is the perturbation on the velocity resulting from the noise on the parameters of the policy. This method lets the user specify the desired norm of the velocity noise vector as unique parameter. It is appropriate when there is no prior knowledge on the task, to make sure that each dimension is equally explored. We used this method for the simulation experiments presented in section 3.3.

It should be noted that a directional bias in the velocity noise vector could be introduced following a similar approach with a different constraint regarding isotropy if knowledge about the task should motivate such choice.

The second method insures stability of the trajectory for every roll-out by shaping the exploration noise such that it satisfies the constraints of SEDS. This method is appropriate when the system should never depart from a given region of the state space. This can be useful when experimenting on real hardware systems which have strict limitations regarding their state. It can also help focus the exploration on a region of the state space which is thought to be promising. However the constraints on the noise can slow down the learning and prevent some regions of the parameter space from being explored, so the method is not always suitable.

5 Learning the Stiffness Profile

So far we have considered only learning of a task model using a time-invariant DS representation. In order to

actually execute a task on the robot we need a controller that makes the robot move according to our DS, and learn the additional parameters that this controller introduces.

5.1 Using DS as a Trajectory Generator

The input to the learned model is position and the output is velocity. As we perform a task that requires physical contact, we use an impedance control framework that uses the DS as reference trajectory. This allows us to regulate the physical interaction dynamics via the specification of impedance parameters, which may vary during task execution. Impedance control requires a reference position, which we obtain by integrating the output from our DS. It should be pointed out that in doing so, our controller is no longer time-invariant and we hence loose some of its robustness properties to perturbations on the robot structure. Robustness w.r.t perturbations of the reference frame of the DS is still guaranteed, which can be exploited e.g. to converge to a moving target point, see [7].

While integration of the DS removes the possibility of responding to physical perturbations of the robot arm, we compensated for this by allowing a variable stiffness profile, which allows tuning the way the robot responds to external forces during the task. In the next section we detail how the robot learns this varying stiffness profile in addition to its DS model.

It should be noted that our architecture is one possibility on how to use a time-invariant DS model at task execution time. We recently proposed an alternative architecture [45], which gets rid of the stiffness term completely and is solely based on velocity feedback control. The approach taken in this paper has the advantage of adhering more strictly to the well known impedance control framework, and can be implemented on any platform supporting the same. The controller in [45] requires a torque control interface, but can be a better choice in situations were interaction with the environment is prone to destabilizing the system. Hence, which one to use depends on the application, and is covered in greater detail in [45].

5.2 Variable impedance control architecture

We consider here the full system composed of:

1. The system that needs to track the trajectory (here a simple point mass with some damping):

$$\boldsymbol{x}_{t_i} = \boldsymbol{x}_{t_{i-1}} + \boldsymbol{v}_{t_i} dt\,, \tag{29}$$
$$\boldsymbol{v}_{t_i} = \boldsymbol{v}_{t_{i-1}} + \boldsymbol{a}_{t_i} dt\,, \tag{30}$$
$$\boldsymbol{a}_{ti} = (\boldsymbol{u}_{t_i} - \boldsymbol{v}_{t_{i-1}} \cdot damping)/mass\,, \tag{31}$$

   where $\boldsymbol{x}_{t_i}$ is the position, $\boldsymbol{v}_{t_i}$ the velocity and $\boldsymbol{a}_{t_i}$ the acceleration of the point mass at time t_i. The command $\boldsymbol{u}_{t_i}$ comes from the controller.
2. The reference trajectory :

$$\boldsymbol{x}^{ref}_{t_i} = \boldsymbol{x}^{ref}_{t_{i-1}} + \boldsymbol{v}^{ref}_{t_i} dt\,, \tag{32}$$

   obtained by discrete integration of the reference velocity $\boldsymbol{v}^{ref}_t$, which comes from the GMR parameterized policy (see Equations 1 to 4).
3. The impedance controller :

$$\boldsymbol{u}_{t_i} = \boldsymbol{S}_{t_i}(\boldsymbol{x}^{ref}_{t_{i-1}} - \boldsymbol{x}_{t_i}) + \boldsymbol{D}_{t_i}(\boldsymbol{v}^{ref}_{t_{i-1}} - \boldsymbol{v}_{t_i})\,, \tag{33}$$

   where $\boldsymbol{S}_{t_i}$ and $\boldsymbol{D}_{t_i}$ are respectively the stiffness and damping matrices at time t_i.

In variable impedance control, the stiffness matrix $\boldsymbol{S}$ and damping matrix $\boldsymbol{D}$ are variable matrices. The sequence of values that the stiffness and damping matrix take along the trajectory is what we call the stiffness and damping profiles.

In our architecture, we use a time-invariant DS to model the stiffness profile, just as we do for the reference trajectory. The stiffness matrix is thus a state dependent matrix.

We choose to use a diagonal stiffness matrix $\boldsymbol{S} = \text{diag}(\boldsymbol{s})$ to reduce the number of parameters to be learned. We also choose to directly compute the damping matrix from the stiffness as $\boldsymbol{D} = \text{diag}(2\sqrt{\boldsymbol{s}m})$ to keep the system critically damped.

5.3 The RL formulation

We now show how the stiffness profile for the impedance controller can be learned through Reinforcement Learning at the same time as the reference trajectory.

Our GMR parameterization of the policy allows us to integrate the stiffness parameter $\boldsymbol{s}$ as an additional output of the policy. This is done by extending our GMM to represent the joint probability $p(\boldsymbol{x}^{ref}, \boldsymbol{v}^{ref}, \boldsymbol{s})$ of the position, velocity and stiffness for our set of demonstrations. Stiffness data for demonstrations of the task can be obtained for example by using the method presented in [17]. In the case where no stiffness data is available, it is possible to set the stiffness of the demonstrations to a default uniform value in order to initialize the stiffness to this value.

Just like the desired velocity, the stiffness is queried at every time step using the GMR given the current reference position:

$$\begin{bmatrix} \boldsymbol{v}^{ref} \\ \boldsymbol{s} \end{bmatrix} = GMR(\boldsymbol{x}^{ref}; \boldsymbol{\theta}) \,. \tag{34}$$

When the stiffness profile is integrated to the GMR, the matrix $\boldsymbol{G}$ from Eq. 8 is augmented for each kernel by Dim_{stiff} raws and by $Dim_{stiff} \cdot (Dim_{in} + 1)$ columns, where Dim_{in} is the number of dimensions of the input to the GMR and Dim_{stiff} is the number of dimensions of the stiffness. Note that the dimension of the stiffness and of the trajectory can be different if several dimensions of the trajectory use the same stiffness. The parameter vector $\boldsymbol{\theta}$ from Eq. 10 is augmented for each kernel by $Dim_{stiff} \cdot (Dim_{in} + 1)$. More details can be found in Appendix C.1.

As explained in [16], learning the parameters of a controller (here the stiffness parameter vector $\boldsymbol{s}$) using PI^2 can be justified by considering that these parameters are variables that are also governed by ODEs with very small time constants. More details can be found in Appendix C.2.

Exploration noise is applied to the parameters that act on the stiffness profile the same way as for the other parameters of the policy, meaning that fixed noise terms are drawn from a Gaussian distribution for each roll-out. The variance of the noise for each parameter is hand tuned.

5.4 Simulation results

Two simulation experiments have been carried out to test the performance of the algorithm.

In the first experiment, the task is to learn a motion in a divergent force field [2]. The trajectory has to pass through some via-points while the stiffness has to be maintained as low as possible. This represents the fact that we want to keep the system as compliant as possible for safety reasons. The initial stiffness profile is at a constant level of $\boldsymbol{s}^0 = [20 \quad 20]^T$, which is quite high given the force level. The successful completion of the task requires a refining of the reference trajectory together with the tuning of the stiffness to a minimal level that still ensures stable tracking. It is thus a good test scenario for the simultaneous learning of the reference trajectory and stiffness profile. Details of the simulation can be found in Appendix C.3.

[2] Note that the divergent force field is a perturbation that has nothing to do with the exploration noise. The divergent force field is not stochastic, and it is part of the system.

Figure 9 shows the trajectory resulting from the initial policy learned from demonstration. The reference trajectory does not exactly follow the nominal trajectory and the actual trajectory is thus disturbed by the force field despite the initial high stiffness. Note that the trajectory starts a little bit off the nominal zero perturbation force path to increase the challenge.

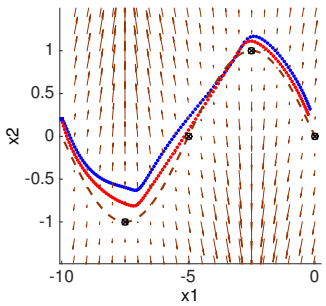


Fig. 9 First experiment: Initial reference trajectory (red) and actual trajectory (blue), starting from the blue square. The arrows represent the force field and the dashed line is the nominal path where the perturbation is equal to zero. The via-points are shown as black crossed circles.

Figure 10 shows the trajectories resulting from the final policies after 100 RL iterations (using 10 roll-outs each) for 20 runs of the experiment. The maps of the mean state dependent stiffness learned through RL are shown, as well as the actual mean stiffness followed during the trajectories. It can be seen that despite the decrease in stiffness that happened during the RL, most of the trajectories pass more accurately through the via-points than with the initial policy. Both components of the variable impedance control have thus been successfully learned simultaneously.

Figure 11 shows the mean learning curve for the 20 runs of the experiment. It can be seen that both the cost related to the stiffness and the cost related to the accuracy of passing through the via-points decreases significantly.

In the second experiment, the task is again to pass through the same set of via-points with a penalty for high stiffness as in the first experiment (cost function can be found in Appendix C.3), but the disturbance force is different. Here the disturbance force has a constant value $\boldsymbol{f} = [0 \quad 10]^T$ but is only present in the half plane $x_1 > -5$. Moreover, the disturbance force is only present in 50% of the executions of the task. The rest of the time, there is no disturbance at all. The goal is to learn a policy that yields the lowest cost on average.

In this setting, the stiffness needs to be increased in the part of the state space where the disturbance might be present, because the initial stiffness $\boldsymbol{s}^0 = [10 \quad 10]^T$ is not sufficient to counter the disturbance. Because it

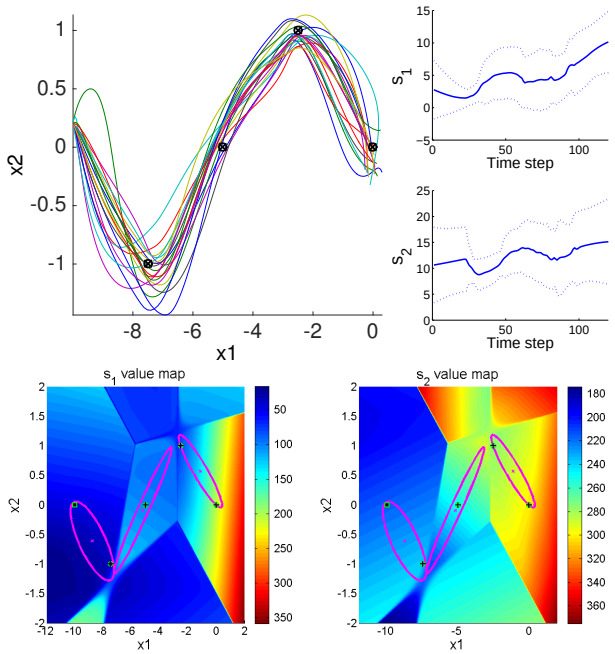


Fig. 10 Top Left: final (actual) trajectories for the 20 runs of the first experiment, with the via-points shown as black crossed circles. Top Right: mean of the final stiffness profile followed during the trajectory (on top in the 1st dimension, bottom in the 2nd dimension) Bottom: map of the state dependent stiffness (left in 1st dimension, right in 2nd dimension), with the basis functions (magenta ellipses) and via-points (green circles with black squares).

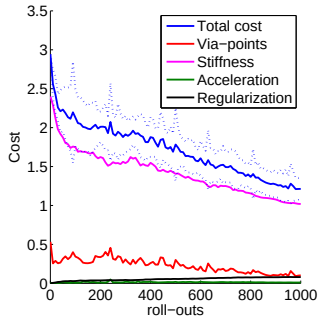


Fig. 11 Mean of the learning curve across the 20 runs of the first experiment. Both the cost due to the stiffness and to the via-points decrease.

is not always present, the disturbance cannot be countered simply by shifting the reference trajectory. The stiffness also needs to be decreased in the part of the state space where no disturbance is expected, as there is a penalty for high stiffness.

In order to learn efficiently in these conditions, we need to modify slightly the learning protocol. Each new policy is evaluated against each situation (with and without disturbance), and the cost of the roll-out is the average of the two trials. This method allows to disambiguate between the role of the reference trajectory and the stiffness.

Note that as every roll-out sees each situation, the system remains deterministic, with the only stochasticity being the exploration noise. Having the two trials with and without disturbance can be seen as equivalent to increasing the the number of states of the system to represent two trajectories simultaneously.

Figure 12 shows the initial trajectories generated by the initial policy with and without the disturbance force. The reference trajectory learned from demonstrations is already close to the via-points. Without the disturbance force, the actual trajectory tracks the reference accurately. However when the disturbance force is present, the actual trajectory is deviated far away from the two last via-points.

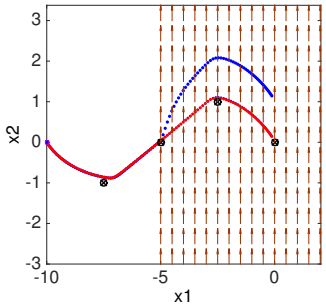


Fig. 12 Second experiment: initial reference trajectory (red) and actual trajectories (blue) with and without disturbance, starting from the blue square. The actual trajectory without disturbance tracks almost perfectly the reference trajectory (and is thus virtually invisible in the graph), while the disturbed actual trajectory is deviated far from the two last via-points.

Figure 13 shows a typical examples of the trajectories (with and without perturbation) resulting from the final policy after 100 RL iterations. The maps of the mean state depended stiffness learned through RL are shown, as well as the actual mean stiffness followed during the trajectories. It can be seen that the reference trajectory has been slightly modified to pass more precisely through the via-points. The perturbed trajectory is also less deviated than initially. This can be explained by the fact that in the final policy, the stiffness in the x_2 dimension has increased a lot in the second part of the trajectory where the perturbation can occur. On the contrary it has decreased in the initial part of the trajectory, and it has decreased everywhere it the x_1 dimension.

Figure 14a shows the mean learning curve over 20 runs of the experiment. The learning curve exhibits a fast decrease in the via-point cost in the initial iterations, while the stiffness cost increases slightly. After that, the via-point decreases at a slower rate, but the stiffness cost starts decreasing slowly as well. These two

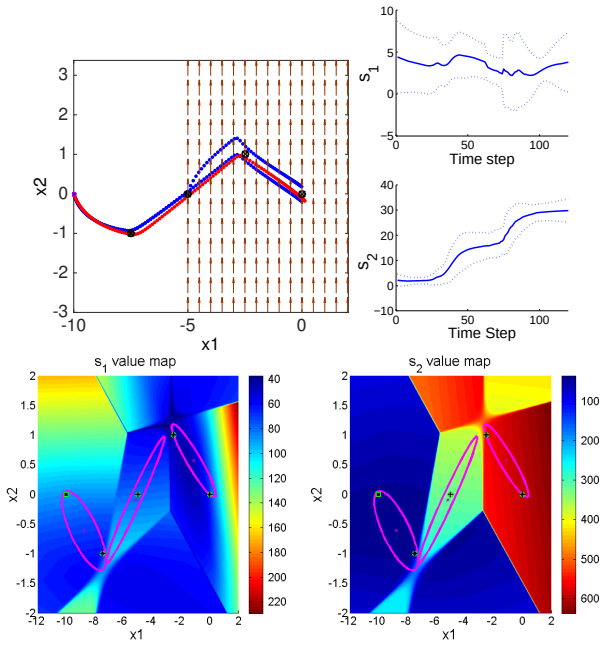
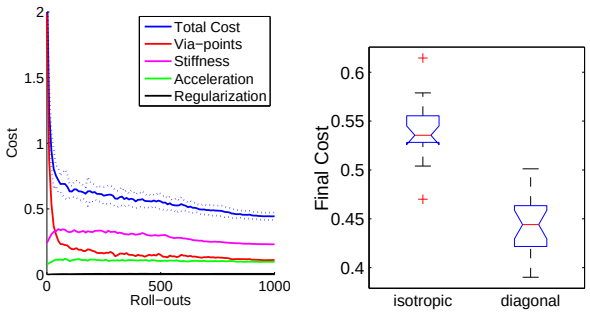


Fig. 13 Top Left: Typical final trajectories (with and without disturbance force) Top Right: mean of the final stiffness profile followed during the trajectory (on top in the 1st dimension, bottom in the 2nd dimension) Bottom: map of the state dependent stiffness (left in 1st dimension, right in 2nd dimension), with the basis functions (magenta ellipses) and via-points (green circles with black squares).

phases can be interpreted as a initial global stiffening of the controller to counter the perturbation, after what the stiffness profile starts being refined in the different parts of the trajectory and in the different dimensions.



(a) (b)

Fig. 14 Left: Mean of the learning curve across the 20 runs of the second experiment. Both the cost due to the stiffness and to the via-points decrease. **Right:** Comparison of the final cost across 20 runs for a system with one single stiffness parameter (isotropic) and one stiffness parameter per dimension (diagonal). This result shows that the algorithms is capable of exploiting the potential of a diagonal stiffness matrix.

It is interesting to compare the learning potential of a system with one single stiffness parameter (isotropic stiffness) to a system with one stiffness parameter per dimension (diagonal stiffness, as considered so far). Figure 14b presents a boxplot comparison of the final total cost for the two systems. It can be seen that the system with different stiffness parameters for each dimension performs significantly better. We can conclude that the learning algorithm is able to leverage the greater flexibility of the more complex model.

6 Experimental results

A hardware experiment was designed to validate the proposed algorithm with simultaneous learning of the reference trajectory and of the stiffness profile. The results are presented and analyzed in this section.

The hardware experiment is carried on a KUKA LWR robot. It is a 7-DOF robotic arm. The test scenario consists of the task of shoveling gravel from a large box. It is a planar task involving only two positional variables and one angle variable in the Cartesian reference frame, as depicted on Figure 15. A shovel instrumented with a 6 axis force/torque sensor is held firmly by a BarrettHand mounted at the end of the LWR.

The objective of the task is to get as much gravel as possible on the shovel within a given time (15 seconds), while avoiding high interaction forces with the environment and while remaining as compliant as possible. This objective is representative of situations where a certain task must be carried out as efficiently as possible while taking into account safety constraints. Additional penalties are given for high accelerations during the task and for not reaching a given goal at the end of the trajectory. These penalties do not represent the primary goal of the task but are necessary to prevent impractical behaviors (e.g. excessive contact forces) from occurring.

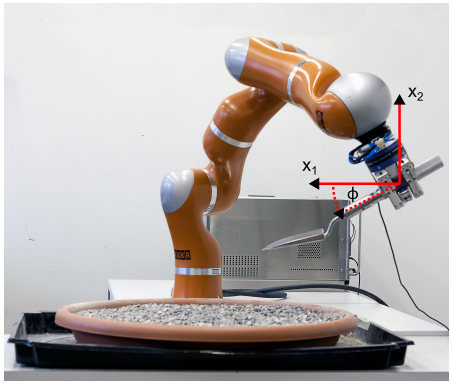


Fig. 15 Setting of the hardware experiment with the KUKA LWR robotic arm. The task is planar and the state of the end effector is determined by its position $[x_1, x_2]$ and orientation ϕ.

The cost function for this task is given by the following terminal cost Φ_r, immediate cost $q_{t_i,r}$ and control cost matrix $\boldsymbol{R}$ that define the cost function according to Eq. 6:

$$\begin{aligned}\Phi_r &= (f^{WeightGravel} - 3.5) + 10 \cdot ||\boldsymbol{\xi}^{goal} - \boldsymbol{\xi}_{t_{N_t}}||^2 \,, \\ q_{t_i,r} &= 10^{-6}||\tilde{\boldsymbol{F}}_{t_i}||^2 + 5 \cdot 10^{-7}||\boldsymbol{s}_{t_i}|| + 10^{-4}||\dot{\boldsymbol{\xi}}_{t_i,r} - \dot{\boldsymbol{\xi}}_{t_{i-1},r}||^2 \,, \\ \boldsymbol{R} &= 10^{-12} \cdot I \,,\end{aligned} \tag{35}$$

where $\boldsymbol{\xi}_{t_i} = [x^1_{t_i} \quad x^2_{t_i} \quad \phi_{t_i}]^T$ is a vector containing the Cartesian position and angle of the end effector of the robot in the task plane measured at time t_i, $f^{WeightGravel}$ is the force measured at the end of the trajectory along the negative x_2 axis, and $\tilde{\boldsymbol{F}}_{t_i}$ is the force vector measured at time t_i. The latter is corrected so that forces due to the weight of the gravel are removed.

The control architecture is similar to the one described in Section 5 but the controlled system is now the robotic arm. The robot is gravity compensated and is controlled with a Cartesian impedance controller that takes as input the desired Cartesian position $\boldsymbol{x}$ and angle $\boldsymbol{\phi}$ in the task plan, as well as scalar stiffness parameter s. The position and orientation outside the task plane are fixed. The same stiffness is used for all axis, with a scaling factor of 10 for the orientation stiffness.

The inputs to the Cartesian impedance controller come from the parameterized policy. The stiffness is directly output by the GMR while the desired position $\boldsymbol{x}^{ref}$ and angle ψ come from integration of the desired velocity $\boldsymbol{v}^{ref}$ and angular rate ω^{ref}:

$$\begin{bmatrix} \boldsymbol{v}^{ref}_{t_i} \\ \omega^{ref}_{t_i} \\ s_{t_i} \end{bmatrix} = GMR(\boldsymbol{x}^{ref}_{t_{i-1}}; \boldsymbol{\theta}) \,, \tag{36}$$

$$\boldsymbol{x}^{ref}_{t_i} = \boldsymbol{x}^{ref}_{t_{i-1}} + \boldsymbol{v}^{ref}_{t_i} dt \,, \tag{37}$$

$$\phi^{ref}_{t_i} = \phi^{ref}_{t_{i-1}} + \omega^{ref}_{t_i} dt \,. \tag{38}$$

Note that here the GMR takes as input only the position and not the angle. This choice reduces the number of parameters to be learned by reducing the number of inputs, while still offering enough flexibility to learn the task.

The learning is done as follows. An initial policy was learned with a GMM with four components ($N_G = 4$ in Eq. (1)) from a batch of five demonstrations of the task. The demonstrations were given by a human backdriving the robot and shoveling some soil. The initial stiffness profile is set to a uniform value s^0. The initial policy is most probably sub-optimal as the demonstrations were given in another material (soil vs. gravel) and because the stiffness profile was chosen to a constant default value. The RL algorithm was applied for ten iterations using eight rollouts each, with the nominal policy from the previous iteration used as one of the rollouts[3]. The homogeneous exploration scheme from subsection 4.1 was used to generate the exploration noise in this experiment.

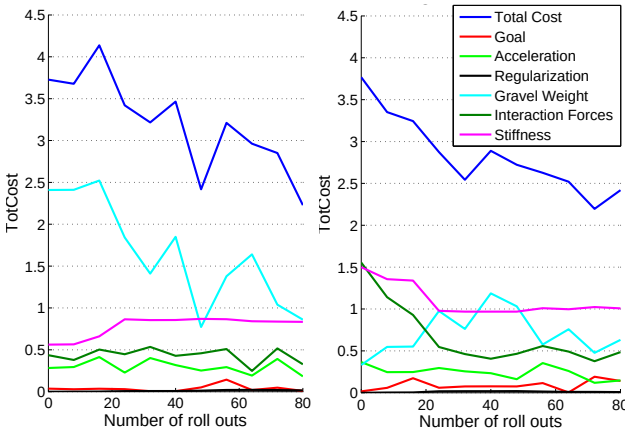


Fig. 16 Learning curves for the two hardware experiments. The two hardware experiments were initialized with the same trajectory and with **Left:** initial stiffness $s^0 = 150$ N/m. **Right:** initial stiffness $s^0 = 450$ N/m.

Two experiments were performed with different values for the initial uniform stiffness s^0. The learning curves of the experiments are shown in Figure 16 . In the first experiment, the initial stiffness is set to a low value ($s^0 = 150$ N/m) which results in small interaction forces but poor ability to penetrate the gravel. This can be seen by the individual cost terms for collected gravel and interaction force in Fig. 16, left. During the first updates of the policy, the mean of the stiffness increases, as can be seen from the increase in the associated penalty. This results in a better penetration and in more gravel being shoveled. The interaction forces do not really increase, suggesting that the trajectory is also modified and becomes more adapted to the task.

In the second experiment, the initial stiffness is set to a high value ($s^0 = 400$ N/m) which allows the robot to follow the reference trajectory accurately and to shovel a large payload of gravel in the initial rollouts. However, the large interaction forces during the shoveling and the high stiffness of the robot cause large penalties. During the first update of the policy, the mean of the stiffness decreases and so do the interaction forces. However, the quantity of gravel shoveled decreases. But

[3] This method can improve the robustness by preventing problematic situations where none of the roll-outs yield good performance.

after the mean stiffness stabilizes halfway through the learning process, the quantity of gravel shoveled improves again, suggesting that in the second half the trajectory and stiffness profile are fine tuned and become more efficient.

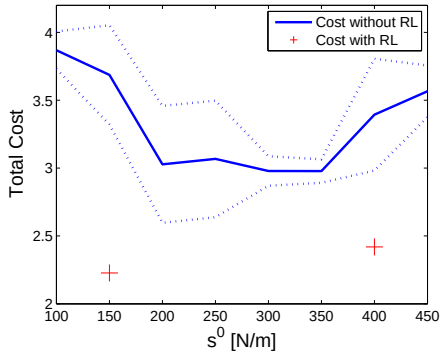


Fig. 17 Total Cost resulting from policies learned from the same demonstration set (same resulting reference trajectory) but with different uniform stiffness s^0. The average over 5 trials is reported with the 1 standard deviation envelope. The two red crosses show the final total cost obtained after 80 rollouts starting with a uniform stiffness of 150 (first experiment) and 400 (second experiment).

The effectiveness of the proposed algorithm is demonstrated by comparing the cost of the policies refined by reinforcement learning with the cost of the policies learned only form the demonstration set and with a constant stiffness value. Figure 17 shows the average total cost of policies which are all learned from the same set of demonstrations (same resulting reference trajectories) but have a different initial uniform stiffness s^0. It can be seen that the lowest cost occurs for uniform stiffness between 200 and 350 N/m. However the cost is still larger for any value of s^0 than the cost obtained with the policies refined by our algorithm. This shows that it is useful to have a variable stiffness and to adapt the trajectory simultaneously, and that our algorithm can achieve this efficiently.

7 Conclusion and Future Work

In this paper we have shown how a variation of the PI^2 policy search RL algorithm can be applied to time-invariant parameterized policies representing non-linear time-invariant DS using GMR. We have formulated constraints on the exploration noise that enable the algorithm to be used with SEDS parameterized policies as well, providing globally asymptotically stable policies. We further extended our parameterized policy so that it can represent a state-dependent stiffness profile in addition to the reference trajectory, which allows to use the learned policies in a variable impedance control architecture. We demonstrated our approach in set of simulated tasks as well as on a challenging digging task.

The most important contributions of this paper are 1) combining the advantages of time-invariant motion representation and PI^2 reinforcement learning (Section 3), 2) development of exploration techniques that can facilitate learning and crucially ensure stable exploration (Section 4) and 3) simultaneous learning of DS and *state-dependent* varying stiffness model. This work brings LfD and RL together by using demonstrations of the task to build an initial trajectory. This procedure provides us, in addition to initial values for the learned parameters, with non-linear features for the parameterized policy that are relevant to the task at hand.

Our time-invariant motion representation is best suited for manipulation tasks usually with $D < 7$, but it also generalizes to higher dimension spaces if necessary [12, 7]. Of course, as more degrees of freedom are considered, the number of parameters to be adapted by RL increases, usually with a negative impact on rate of convergence. That said, PI^2 has been demonstrated to work well even with a large number of parameters [14].

When using a time-invariant DS model for describing a robot task one has to make a choice at task execution time on how to convert the output from the DS into control commands for the robot. In this work, we followed our previous approach from [17] and integrated a trajectory from a given starting point. That trajectory was then used with an impedance controller with a learned, varying stiffness profile. This means that physical perturbations of the robot will not affect the nominal trajectory during task execution. Instead, we allow the robot to deal appropriately with physical perturbations via the stiffness design, which was given by a the teacher in [17] and learned autonomously in this paper.

In order to apply PI^2 to our time-invariant DS formulation, it was necessary to freeze the non-linear features (locations of Gaussians in position space) during policy improvement. This has the drawback that we must rely on receiving relatively good demonstrations in order for the Gaussian basis functions to be placed in locations that are relevant to the task. Naturally, the ability to improve the policy is limited by not being able to recruit expressive power in parts of the input domain that were not seen in the demonstrations. Hence, in order for RL to introduce more significant changes to the motion, it may be required to add new Gaussians in the relevant part of the task space. We consider this the natural direction for future work on this topic.

There is also interesting further work to be done on the topic of exploration. This paper introduced two methods that shape the exploration noise for specific exploration strategies. We think that these methods can be further improved by using the data from the demonstration to tune the exploration noise. Furthermore, recent approaches of PI^2 have shown that there are benefits to modifying the exploration noise during the learning by updating its covariance matrix [15], [46]. Bringing the benefits of these these methods and our methods together is also a interesting direction for improvement.

We would like to add a final remark about some intrinsic limitations of PI^2 that have an influence on the capabilities of the proposed approach. One limitation is that PI^2 assumes that there is no noise in the system apart from the exploration noise. This means that perturbations encountered during the sampled trajectories have a negative effect on the learning and cannot be taken into account to improve the policy. The second limitation is that PI^2 was initially designed to learn trajectories from one specific initial state. Using it to learn trajectories from multiple initial states increases the number of required roll-outs. With these limitations, the learning of policies that generalize well on large areas of the state space remains challenging. Recent developments in the field of path integrals might bring solutions to improve our method further. One work of particular interest is [47], which presents an approach that leverages the perturbations experiences during trials to learn a robust policy with state feedback.

8 Acknowledgment

This research was funded by the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement no 288533 ROBOHOW.COG and by the Swiss National Science Foundation through the National Center of Competence in Research Robotics.

References

1. A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Handbook of Robotics Chapter 59: Robot Programming by Demonstration," in *Handbook of Robotics*. Springer, 2008.
2. S. Schaal, A. Ijspeert, and A. Billard, "Computational approaches to motor learning by imitation." *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, vol. 358, no. 1431, pp. 537–47, mar 2003.
3. R. S. Sutton and A. G. Barto, "Reinforcement Learning," *Learning*, vol. 9, no. 1, pp. 1–23, 1998.
4. J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, pp. 1238–1274, 2013.
5. S. Calinon, P. Kormushev, and D. G. Caldwell, "Compliant skills acquisition and multi-optima policy search with EM-based reinforcement learning," *Robotics and Autonomous Systems*, vol. 61, no. 4, pp. 369–379, Apr. 2013.
6. J. Kober and J. Peters, "Imitation and Reinforcement Learning," *IEEE Robotics Automation Magazine*, vol. 17, no. 2, pp. 55–62, 2010.
7. S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with gaussian mixture models," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 943–957, Oct. 2011.
8. A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, vol. 2. IEEE, 2002, pp. 1398–1403.
9. F. Farshidian, M. Neunert, and J. Buchli, "Learning of closed-loop motion control," in *IEEE International Conference on Intelligent Robots and Systems*, no. IROS, 2014, pp. 1441–1446.
10. F. Guenter, M. Hersch, S. Calinon, and A. Billard, "Reinforcement learning for imitating constrained reaching movements," *Advanced Robotics*, vol. 21, no. 13, pp. 1521–1544, 2007.
11. C. Daniel, G. Neumann, and J. Peters, "Learning concurrent motor skills in versatile solution spaces," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 3591–3597.
12. E. Gribovskaya, S. M. Khansari-Zadeh, and A. Billard, "Learning non-linear multivariate dynamics of motion in robotic manipulators," *The International Journal of Robotics Research*, p. 0278364910376251, 2010.
13. A. Lemme, K. Neumann, R. Reinhart, and J. Steil, "Neural learning of vector fields for encoding stable dynamical systems," *Neurocomputing*, vol. 141, pp. 3–14, Oct. 2014.
14. E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *The Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, 2010.
15. F. Stulp and O. Sigaud, "Policy improvement methods: Between black-box optimization and episodic reinforcement learning," 2012.
16. J. Buchli, F. Stulp, E. Theodorou, and S. Schaal, "Learning variable impedance control," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 820–833, Jun. 2011.
17. K. Kronander and A. Billard, "Learning compliant manipulation through kinesthetic and tactile human-robot interaction," *Transactions on Haptics*, vol. 7, no. 3, pp. 1–16, 2013.
18. P. Pastor, L. Righetti, M. Kalakrishnan, and S. Schaal, "Online movement adaptation based on previous sensor experiences," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 365–371.
19. J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Machine Learning*, vol. 84, no. 1-2, pp. 171–203, nov 2010.
20. S. Calinon, F. D'halluin, E. L. Sauser, D. G. Caldwell, and A. G. Billard, "Learning and reproduction of gestures by imitation," *Robotics & Automation Magazine, IEEE*, vol. 17, no. 2, pp. 44–54, 2010.
21. H. G. Sung, "Gaussian mixture regression and classification," Ph.D. dissertation, RICE UNIVERSITY, 2004.

22. K. Kronander, S. M. Khansari-Zadeh, and A. Billard, "Incremental motion learning with locally modulated dynamical systems," *Robotics and Autonomous Systems*, vol. 70, pp. 52–62, 2015.
23. S. M. Khansari-Zadeh and A. Billard, "Learning stable non-linear dynamical systems with Gaussian Mixture Models," *IEEE Transactions on Robotics*, vol. 27, pp. 1–15, 2011.
24. J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7-9, pp. 1180–1190, Mar. 2008.
25. J. Kober and J. Peters, "Learning motor primitives for robotics," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 2112–2118.
26. N. Vlassis, M. Toussaint, G. Kontes, and S. Piperidis, "Learning model-free robot control by a monte carlo EM algorithm," *Autonomous Robots*, vol. 27, no. 2, pp. 123–130, Aug. 2009.
27. V. Gullapalli, J. A. Franklin, and H. Benbrahim, "Acquiring robot skills via reinforcement learning," *Control Systems, IEEE*, vol. 14, no. 1, pp. 13–24, 1994.
28. R. Tedrake, T. W. Zhang, and H. S. Seung, "Stochastic policy gradient reinforcement learning on a simple 3d biped," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2849–2854.
29. N. Hogan, "Impedance control: An approach to manipulation," *Journal of Dynamic Systems Measurement and Control*, vol. 107, no. 12, pp. 1–24, 1985.
30. E. Burdet, R. Osu, D. W. Franklin, T. E. Milner, and M. Kawato, "The central nervous system stabilizes unstable dynamics by learning optimal impedance." *Nature*, vol. 414, no. 6862, pp. 446–9, nov 2001.
31. L. P. J. Selen, D. W. Franklin, and D. M. Wolpert, "Impedance control reduces instability that arises from motor noise." *The Journal of neuroscience : the official journal of the Society for Neuroscience*, vol. 29, no. 40, pp. 12 606–16, oct 2009.
32. K. P. Tee, D. W. Franklin, M. Kawato, T. E. Milner, E. Burdet, K. Peng, T. David, and W. F. Mitsuo, "Concurrent adaptation of force and impedance in the redundant muscle system." *Biological cybernetics*, vol. 102, no. 1, pp. 31–44, jan 2010.
33. C. Yang, G. Ganesh, S. Haddadin, S. Parusel, A. Albu-Schaffer, and E. Burdet, "Human-like adaptation of force and impedance in stable and unstable interactions," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 918–930, 2011.
34. M. Howard, D. J. Braun, and S. Vijayakumar, "Transferring Human impedance behavior to heterogeneous variable impedance actuators," *IEEE Transactions on Robotics*, vol. 29, no. 4, pp. 847–862, 2013.
35. A. Ajoudani, N. Tsagarakis, and A. Bicchi, "Tele-impedance: Teleoperation with impedance regulation using a body-machine interface," *The International Journal of Robotics Research*, vol. 31, no. 13, pp. 1642–1656, oct 2012.
36. S. Calinon, I. Sardellitti, and D. Caldwell, "Learning-based control strategy for safe human-robot interaction exploiting task and robot redundancies," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010, pp. 249–254.
37. L. Rozo, S. Calinon, D. Caldwell, P. Jimenez, C. Torras, and P. Jiménez, "Learning collaborative impedance-based robot behaviors," in *AAAI Conference on Artificial Intelligence*, 2013.
38. A. X. Lee, H. Lu, A. Gupta, S. Levine, and P. Abbeel, "Learning Force-Based Manipulation of Deformable Objects from Multiple Demonstrations," *IEEE International Conference on Robotics and Automation*, 2015.
39. M. Toussaint, "Probabilistic inference as a model of planned behavior," *Künstliche Intelligenz*, vol. 3, no. 9, pp. 23–29, 2009.
40. A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic Movement Primitives," *Neural Information Processing Systems*, pp. 1–9, 2013.
41. J. Medina, D. Sieber, and S. Hirche, "Risk-sensitive interaction control in uncertain manipulation tasks," in *IEEE International Conference on Robotics and Automation*, 2013.
42. E. A. Rückert, G. Neumann, M. Toussaint, and W. Maass, "Learned graphical models for probabilistic planning provide a new class of movement primitives," *Frontiers in Computational Neuroscience*, vol. 6, no. January, pp. 1–20, 2013.
43. D. Mitrovic, S. Klanke, and S. Vijayakumar, "Learning Impedance Control of Antagonistic Systems Based on Stochastic Optimization Principles," *The International Journal of Robotics Research*, vol. 30, no. 5, pp. 556–573, 2011.
44. M. Garabini, A. Passaglia, F. Belo, P. Salaris, and A. Bicchi, "Optimality principles in stiffness control: The VSA kick," in *IEEE Intl. Conf. on Robotics and Automation*, no. 1, 2012, pp. 3341–3346.
45. K. Kronander and A. Billard, "Passive Interaction Control With Dynamical Systems," *Robotics and Automation Letters*, vol. 1, no. 1, pp. 106–113, 2016.
46. F. Stulp and O. Sigaud, "Path integral policy improvement with covariance matrix adaptation," *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pp. 281–288, 2012.
47. S. Thijssen and H. Kappen, "Path integral control and state-dependent feedback," *Physical Review E*, vol. 91, no. 3, p. 032104, 2015.
48. S. Calinon, S. Bruno and D.G. Caldwell, "A task-parameterized probabilistic model with minimal intervention control" *IEEE Intl. Conf. on Robotics and Automation*, no. 1, 2014, pp. 3339-3344.

A Additional details for section 3

A.1 Details of the first simulation experiment

The first simulation experiment is performed with $N_r = 10$ and 50 learning iterations. The variance of the exploration noise $\boldsymbol{\Sigma}_\epsilon$ is set to obtain a mean norm of 0.1 for the velocity noise vector (see 4.1).

The cost is composed of the following components:

$$\begin{aligned} \Phi_r &= ||\boldsymbol{x}^{goal} - \boldsymbol{x}_{t_{N_t}}||^2 + \sum_{l=1}^{N_{VP}} \min_{j=1:N_t} ||\boldsymbol{x}_l^{via} - \boldsymbol{x}_{t_j}||^2 \,, \\ q_{t_i,r} &= 0.0001 \cdot ||\dot{\boldsymbol{x}}_{t_i,r} - \dot{\boldsymbol{x}}_{t_{i-1},r}|| \,, \\ \boldsymbol{R} &= 0.0005 \cdot I \,. \end{aligned} \tag{39}$$

At each time step, the immediate cost penalizes the acceleration to promote smoother trajectories. The final cost is the sum of the squared minimum distance to each via-point along the trajectory, plus the squared distance to the goal at the final time step. Unlike the via-points, the goal has to

be reached at a specific time, i.e. at the end of the trajectory. On top of that there is the control cost[4] that penalizes for the size of the change from the initial parameter vector ($\boldsymbol{\theta}^* = \boldsymbol{\theta} - \boldsymbol{\theta}^0$), through the control cost matrix $\boldsymbol{R}$. It is rather low and serves here more as a regularization term that can prevent the parameter vector from becoming very large.

A.2 Details of the second simulation experiment

The second simulation experiment is performed with $N_r = 20$ and 200 learning iterations. The variance of the exploration noise $\boldsymbol{\Sigma}_\epsilon$ is set to obtain a mean norm of 0.1 for the velocity noise vector (see 4.1).

The cost function is similar to the cost functions of the first experiment. The only difference is that for the "via-points" cost term, two sets of via-points are distinguished, one in the right-half plane and on in the left half-plane of the state space. The "via-points" cost that is applied is the minimum between the cost relative to each set. This means that the trajectory only has to pass through one of the sets of via-points. The cost function is thus composed of the following components:

$$\begin{aligned} \Phi_r &= ||\boldsymbol{x}^{goal} - \boldsymbol{x}_{t_{N_t}}||^2 + \min_{b=1:2} \sum_{l=1}^{N_b^{VP}} \min_{j=1:N_t} ||\boldsymbol{x}_{l,b}^{via} - \boldsymbol{x}_{t_j}||^2 \,, \\ q_{t_i,r} &= 0.0002 \cdot ||\dot{\boldsymbol{x}}_{t_i,r} - \dot{\boldsymbol{x}}_{t_{i-1},r}|| \,, \\ \boldsymbol{R} &= 0.0005 \cdot I \,, \end{aligned} \tag{40}$$

where index b is for the via-points set.

B Additional details for section 4

The cost function for for the constrained reaching task is given by the following terminal cost Φ_r, immediate cost $q_{t_i,r}$ and control cost matrix $\boldsymbol{R}$:

$$\begin{aligned} \Phi_r &= 10\, ||\boldsymbol{x}^{goal} - \boldsymbol{x}_{t_{N_t}}||^2 \,, \\ q_{t_i,r} &= 0.01\, ||\dot{\boldsymbol{x}}_{t_i,r} - \dot{\boldsymbol{x}}_{t_{i-1},r}|| + ||\dot{\boldsymbol{x}}_{t_i,r}||\, \Delta t - \frac{8.815}{N_t} \,, \\ \boldsymbol{R} &= 0.001\, I \,. \end{aligned} \tag{41}$$

Penalizing the norm of the velocity of each time step is equivalent to penalizing the length of the trajectory. The subtractive term is there to remove the minimal possible path length, which is 8.815, from the total cost. The norm of the acceleration is penalized to promote smooth trajectories.

C Additional details for section 5

C.1 Augmented control matrix and parameter vector

Here is an example of the form that takes the matrix $\boldsymbol{G}$ and parameter vector $\boldsymbol{\theta}$ when extended with the stiffness profile. This is for a 1-dimensional stiffness profile (i.e same stiffness applied to all dimensions) and a 2-D system:

$$\boldsymbol{G}_{x_t} = \begin{bmatrix} \boldsymbol{G}_{x_t}^1 & \boldsymbol{G}_{x_{t_i}}^2 & \dots & \boldsymbol{G}_{x_t}^{N_G} \end{bmatrix} \tag{42}$$

$$\boldsymbol{G}_{x_t}^k = h_{x_t}^k \begin{bmatrix} x_{1,t} & x_{2,t} & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & x_{1,t} & x_{2,t} & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & x_{1,t} & x_{2,t} & 0 & 0 & 1 \end{bmatrix} \tag{43}$$

$$\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\theta}^1 & \boldsymbol{\theta}^2 & \dots & \boldsymbol{\theta}^{N_G} \end{bmatrix}^T \tag{44}$$

$$\boldsymbol{\theta}^k = \begin{bmatrix} A_{1,1}^k & A_{1,2}^k & A_{2,1}^k & A_{2,2}^k & A_{3,1}^k & A_{3,2}^k & b_1^k & b_2^k & b_3^k \end{bmatrix} \tag{45}$$

The parameters that have to do with $\boldsymbol{s}$ are $A_{3,1}^k$, $A_{3,2}^k$, and b_3^k.

C.2 About learning control parameters with PI^2

Learning the stiffness profile for an impedance controller may seem to depart from the most common application of PI^2, which is to learn policies in the form of reference trajectories. However, it is explicitly mentioned in [14] that the concept of "action" in path integral optimal control has a broad sens and can be a control gain just as well as a desire state.

In order have the stiffness parameters $\boldsymbol{s}$ fit the form imposed by the PI^2 formalism, these parameters must be seen as described by auxiliary ODEs (one per dimension of the stiffness) that are added to the set of ODEs that represent our point-mass system and our reference trajectory, as explained in [16]. The auxiliary ODEs have the following form:

$$\dot{s_j} = \alpha_k ({\boldsymbol{g}_{x^{ref}}^{s_j}}^T (\boldsymbol{\theta}^{s_j} + \boldsymbol{\epsilon}^{s_j}) - s_j) \tag{46}$$

where the index j represents the dimension and in our 2-D case

$$\boldsymbol{\theta}^{s_j} = \begin{bmatrix} A_{3,1}^j & A_{3,2}^j & b_3^j \end{bmatrix}^T \tag{47}$$

$$\boldsymbol{g}_{x^{ref}}^{s_j} = \begin{bmatrix} h_{x^{ref}}^j \cdot (x_1^{ref} & x_2^{ref} & 1) \end{bmatrix}^T \tag{48}$$

The parameter α_k of this auxiliary ODE are chosen very large so that the s_j converges very fast to its final value ${\boldsymbol{g}_{x^{ref}}^{s_j}}^T (\boldsymbol{\theta}^s + \boldsymbol{\epsilon}^{s_j})$, i.e. much faster than the changes in ${\boldsymbol{g}_{x^{ref}}^{s_j}}^T$ (caused by the evolution of $\boldsymbol{x}^{ref}$). We will thus make the assumption that for any practical purpose $s_j = {\boldsymbol{g}_{x^{ref}}^{s_j}}^T (\boldsymbol{\theta}^{s_j} + \boldsymbol{\epsilon}^{s_j})$ and so by learning $\boldsymbol{\theta}^{\boldsymbol{s}} = [\boldsymbol{\theta}^{s_1} \quad \boldsymbol{\theta}^{s_2} \dots]$ we will learn the stiffness profile (as a function of the state).

C.3 Details of the simulations

The cost function for the first task is given by the following terminal cost Φ_r, immediate cost $q_{t_i,r}$ and control cost matrix $\boldsymbol{R}$:

$$\begin{aligned} \Phi_r &= ||\boldsymbol{x}^{goal} - \boldsymbol{x}_{t_{N_t}}||^2 + \sum_{l=1}^{N_{VP}} \min_{j=1:N_t} ||\boldsymbol{x}_l^{via} - \boldsymbol{x}_{t_j}||^2 \\ q_{t_i,r} &= 10^{-4} \cdot ||\dot{\boldsymbol{x}}_{t_i,r} - \dot{\boldsymbol{x}}_{t_{i-1},r}|| + 5 \cdot 10^{-4} ||\boldsymbol{s}_{t_i}|| \\ \boldsymbol{R} &= 10^{-6} \cdot I \end{aligned} \tag{49}$$

[4] Although for our parameterization the size of the parameters is only loosely related to the actual control that will be applied to the system, we keep the conventional name "control cost" for this term.

The divergent force field increases linearly with the distance to the nominal path. It is modeled by the following equation :

$$\boldsymbol{f}_{t_i} = 10 \cdot \min_j ||\boldsymbol{x}_{t_i} - \boldsymbol{x}_j^{nom}|| \tag{50}$$

where $[\boldsymbol{x}_1^{nom}\boldsymbol{x}_2^{nom} \dots \boldsymbol{x}_{N^{nom}}^{nom}]$ is the discrete representation of the nominal path where the force is null (i.e. the ridge of the divergent force field).

The cost function the second task has the same structure as the one of the first experiment, with slightly different weights for the stiffness and acceleration costs:

$$\begin{aligned} \Phi_r &= ||\boldsymbol{x}^{goal} - \boldsymbol{x}_{t_{N_t}}||^2 + \sum_{l=1}^{N_{VP}} \min_{j=1:N_t} ||\boldsymbol{x}_l^{via} - \boldsymbol{x}_{t_j}||^2 \\ q_{t_i,r} &= 10^{-3} \cdot ||\dot{\boldsymbol{x}}_{t_i,r} - \dot{\boldsymbol{x}}_{t_{i-1},r}|| + 2 \cdot 10^{-4} ||\boldsymbol{s}_{t_i}|| \\ \boldsymbol{R} &= 10^{-6} \cdot I \end{aligned} \tag{51}$$