Bayesian Optimization for Policy Search in High-Dimensional Systems via Automatic Domain Selection

Bayesian Optimization for Policy Search in High-Dimensional Systems via Automatic Domain Selection

Lukas P. Fröhlich[1,2], Edgar D. Klenske[1], Christian G. Daniel[1] and Melanie N. Zeilinger[2,*]

***Abstract*— Bayesian Optimization (BO) is an effective method for optimizing expensive-to-evaluate black-box functions with a wide range of applications for example in robotics, system design and parameter optimization. However, scaling BO to problems with large input dimensions (>10) remains an open challenge. In this paper, we propose to leverage results from optimal control to scale BO to higher dimensional control tasks and to reduce the need for manually selecting the optimization domain. The contributions of this paper are twofold: 1) We show how we can make use of a learned dynamics model in combination with a model-based controller to simplify the BO problem by focusing onto the most relevant regions of the optimization domain. 2) Based on (1) we present a method to find an embedding in parameter space that reduces the effective dimensionality of the optimization problem. To evaluate the effectiveness of the proposed approach, we present an experimental evaluation on real hardware, as well as simulated tasks including a 48-dimensional policy for a quadcopter.**

I. INTRODUCTION

Bayesian optimization (BO) is a powerful method for the optimization of black-box functions which are costly to evaluate. One of the great advantages is its sample efficiency, enabling a wide variety of applications, ranging from hyperparameter search for machine learning algorithms [1], medical applications [2], to robotics [3], [4]. Especially for high-dimensional problems, however, the number of function evaluations—experiments on the hardware in many cases—can still be prohibitive. In this paper, we consider BO in the context of direct policy search for systems with continuous state/action space. This offers the possibility of exploiting knowledge about the problem, and rather than considering the objective function as a black-box, we take a gray-box approach.

For many high-dimensional objective functions it is valid to assume some underlying structure, alleviating the curse of dimensionality. One common assumption is that the effective dimensionality of the objective function is lower and lies in a linear subspace of the parameter domain [5], [6]. Another assumption is that the objective has an additive structure and many parameters are uncorrelated [7]. However, finding an appropriate subspace is hard and the effective dimensionality of the objective function is usually not known a-priori.

In addition to the problem of finding a suitable embedding for dimensionality reduction, it is not clear how to set

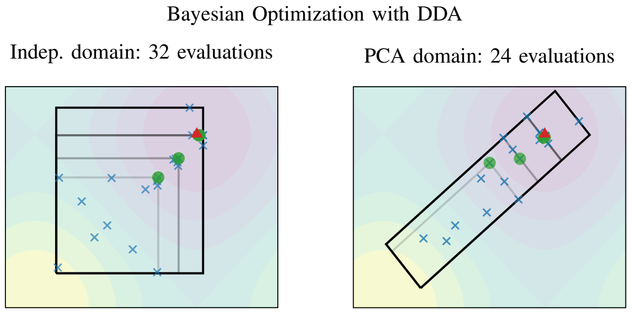


Fig. 1: Illustrative 2D example depicting different optimization domains (see Sec. IV) and the domains' growth due to dynamic domain adaptation (DDA) (see Sec. IV-C), as well as the evaluated points (×). When an estimated optimum (●) is on the domain's boundary (▬), the domain grows in the respective direction. The global optimum is marked by ▲.

the domain boundaries in which the parameters are to be optimized in a meaningful manner. Oftentimes, these boundaries have to be found experimentally or tuned by a domain expert, introducing many additional parameters that significantly influence the convergence of BO. We address both issues by using a learned dynamics model in combination with a model-based technique from optimal control.

The idea of combining BO and methods from optimal control has previously been explored in order to efficiently tune policies for dynamical systems [8], [9]. However, both of these approaches are susceptible to model bias because the respective policy parameterizations depend on a dynamics model. The proposed method in this paper uses a model-based approach only for selecting an appropriate parameter space and initial domain boundaries. The optimization of the policy itself is done in a model-free manner, allowing for higher flexibility and improved final performance.

In particular, our contributions are as follows, see also Fig. 1: 1) Using a learned dynamics model, we show how to automatically select the boundaries of the optimization domain such that no manual tuning or expert knowledge is needed, making BO more widely applicable for the use in policy search. 2) We show how to determine a linear embedding that exploits the structure of the objective function, thus reducing the effective dimensionality of the optimization problem. 3) We propose a scheme to dynamically adapt the domain boundaries during optimization based on the objective function's surrogate model if the initial domain was chosen too small. The scheme to adapt the optimization domain is not limited to the application in policy search, but can be used as

[1]Bosch Center for Artificial Intelligence, Robert Bosch GmbH, 71272 Renningen, Germany. Corresponding author: lukas.froehlich@de.bosch.com

[2]Institute for Dynamic Systems and Control, ETH Zurich, 8092 Zurich, Switzerland

*The research of Melanie N. Zeilinger was supported by the Swiss National Science Foundation under grant no. PP00P2 157601/1.

a general extension to BO. These contributions enable direct policy search based on BO for systems with significantly higher dimensionality than reported in the literature.

II. RELATED WORK

In recent years, BO has emerged as a promising policy search method. In [9], a stochastic optimal control problem is considered and BO is applied to optimize the entries of the linear system matrices that describe the system dynamics, which are then used to construct an linear quadratic regulator (LQR) controller. Similarly, [8] also uses an LQR approach, but instead of tuning the model matrices, the weight matrices of the quadratic cost function are optimized. In contrast to the two aforementioned papers, [3] directly optimizes the entries of a linear state feedback policy. The different parameterizations of these methods are reviewed in Sec. III-C.

Besides linear state feedback policies, other parameterizations have been proposed in the context of BO for policy search. Neural networks have been used in order to learn a swing-up of a double cart-pole system in simulation [10]. On average more than 200 iterations are required to learn a good policy, although no noise is present in the system. In [11], a high-fidelity simulator of a bipedal robot is used in combination with a neural network to learn a tailored kernel for the Gaussian process (GP) surrogate model in BO. To increase the efficiency of BO, [12] proposes to use trajectory data generated during experiments by using a so-called behavior-based kernel, which compares policies by the similarity of their resulting trajectories on the system. However, this approach is limited to stochastic policies. In all methods discussed above, the domain over which the parameters are optimized is always chosen manually, indicating expert knowledge or experience from other experiments. The method proposed in this paper, in contrast, reduces the need for prior information and manual tuning.

One of the most prominent approaches for finding a lower dimensional space in which to perform the optimization is proposed in [5]. Here, the authors assume that the effective dimensionality of the objective function is small and a randomly sampled linear mapping is used to transform the high-dimensional input parameters to the lower dimensional space. However, the effective dimensionality of the subspace is usually not known a-priori and the choice of an appropriate optimization domain is still an issue. Instead of randomly sampling an embedding, it has been proposed to actively learn an embedding [6]. However, the proposed method has not been used in the context of BO, but rather in the active learning setting for GP regression. Thus, evaluation points are not selected according to optimizing an objective, instead they are selected according to their information gain w.r.t. the belief of the embedding itself.

The remainder of this paper is structured as follows: In Sec. III we formally state the policy search problem and review BO. Furthermore, we explain the different parameterizations for linear state feedback policies that have been proposed in the literature. In Sec. IV we describe the contributions of this paper. Results from the simulations and hardware experiments are then discussed in Sec. V.

III. PRELIMINARIES

In this section we formally state the policy search problem and explain how BO can be employed to solve it. Furthermore, we review different parameterizations for linear feedback policies.

A. Policy Search

Consider the problem of finding a policy, $\pi_\theta : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$, mapping the state $\boldsymbol{x}$ to an input $\boldsymbol{u} = \pi_\theta(\boldsymbol{x})$, which is parameterized by $\boldsymbol{\theta} \in \boldsymbol{\Theta} \subset \mathbb{R}^{n_\theta}$, with the goal of minimizing a specified performance criterion, or cost function, J, over a fixed time horizon. Formally, this is defined in the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \sum_{t=0}^{T} \mathbb{E}\left[c(\boldsymbol{x}_t, \pi_\theta(\boldsymbol{x}_t))\right], \\ \text{s.t. } \boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \pi_\theta(\boldsymbol{x}_t)) + \boldsymbol{\nu}, \end{aligned} \tag{1}$$

where $c(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is the cost of being in state $\boldsymbol{x}_t$ and applying input $\boldsymbol{u}_t$, and $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ denotes the (generally unknown) state transition model governing the dynamics of the system that are corrupted by white noise, $\boldsymbol{\nu} \sim \mathcal{N}(0, \boldsymbol{\Sigma}_\nu)$. In this paper, we apply BO to find the parameters $\boldsymbol{\theta}^*$ of a cost-minimizing policy.

B. Bayesian Optimization for Policy Search

In BO, GP regression (see, e.g., [13]) is used to model the performance of the system, $J(\boldsymbol{\theta})$, as a function of the policy parameters $\boldsymbol{\theta}$, based on noisy observations $\hat{J}(\boldsymbol{\theta})$. In the considered setting, one function evaluation corresponds to a rollout of the current policy on the system, after which the new data is added: $\mathcal{D}_{n+1} = \mathcal{D}_n \cup (\boldsymbol{\theta}_{n+1}, \hat{J}(\boldsymbol{\theta}_{n+1}))$. As the objective is expensive to evaluate, the goal is to only use few function evaluations to find the minimum of the cost. After each experiment, a new evaluation point in the domain is selected by maximizing an acquisition function, $\alpha(\boldsymbol{\theta}; \mathcal{D}_n)$. Different acquisition functions have been proposed in the literature, such as probability of improvement (PI) [14], expected improvement (EI) [15], and upper confidence bound (UCB) [16], [17]. They all have in common that they trade off between exploration (i.e., favoring regions of the domain where the objective function has not been evaluated yet) and exploitation (i.e., proposing the estimated optimum of the objective function). For a thorough introduction to BO, we refer the reader to [18].

C. Linear Policy Parameterization

In this paper, we consider linear state feedback policies of the form:

$$\pi_\theta(\boldsymbol{x}) = -\boldsymbol{K}(\boldsymbol{\theta})\boldsymbol{x}. \tag{2}$$

The advantage of linear policies is their low dimensionality compared with other parameterizations, such as (deep) neural networks that need large amounts of training data. As every experiment on real hardware costs considerable amount of time (and potentially money), it is infeasible to perform hundreds or even thousands of rollouts. Another key benefit of a linear policy is that we can leverage the relation to linear optimal controllers to increase the efficiency of BO. Although linear policies are simple in their form, they have shown impressive results, even on complex tasks, such as locomotion [19].

To improve the efficiency of BO, we make use of the LQR, a method commonly applied in optimal control. One way of approximating the problem in Eq. (1) is to linearize the system dynamics, $f(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t$, and quadratize the stage cost, $c(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx \boldsymbol{x}_t^T \boldsymbol{Q} \boldsymbol{x}_t + \boldsymbol{u}_t^T \boldsymbol{R} \boldsymbol{u}_t$. Using these approximations, one can construct the static LQR feedback gain matrix efficiently [20, §6.1]), which we denote as $\boldsymbol{K} = \texttt{dlqr}\,(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{Q}, \boldsymbol{R})$. In fact, if the true system dynamics are linear and the stage cost is quadratic, the static LQR gain matrix is optimal for the case of an infinite time horizon, i.e., $T \to \infty$. However, it leads to suboptimal performance in the case of a nonlinear system as considered in this paper. In the following, we review three different parameterizations for linear policies in order to apply BO-based policy search, two of which make use of the LQR.

1) Optimizing the Gain Matrix: In [3], BO is applied by directly optimizing the feedback gain matrix, i.e., each entry in $\boldsymbol{K}^K(\boldsymbol{\theta})$ corresponds to one optimization parameter. Thus, the number of parameters scales linearly in the number of states and inputs. This method is model-free, i.e., it does not make use of the LQR, and thus no (linear) dynamics model is required for this parameterization.

2) Optimizing System Matrices: This particular parameterization was first used in [9]. The idea is to parameterize the system matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, where each entry of the matrices corresponds to one parameter, from which then the respective LQR can be calculated:

$$\boldsymbol{K}^{AB}(\boldsymbol{\theta}) = \texttt{dlqr}\,(\boldsymbol{A}(\boldsymbol{\theta}), \boldsymbol{B}(\boldsymbol{\theta}), \boldsymbol{Q}, \boldsymbol{R})\,. \tag{3}$$

With this parameterization, a task-specific model is learned, which can be better than, e.g., the true model linearized around an operating point. However, the number of parameters scales quadratically with the number of states, thus making this approach infeasible for large state spaces.

3) Optimizing Weight Matrices: Given a linear approximation of the dynamics, this method tunes the LQR's weight matrices instead of the system matrices [8]:

$$\boldsymbol{K}^{QR}(\boldsymbol{\theta}) = \texttt{dlqr}\,(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{Q}(\boldsymbol{\theta}), \boldsymbol{R}(\boldsymbol{\theta}))\,. \tag{4}$$

Commonly, only the diagonal entries of the weight matrices are tuned, i.e., the matrices are of the following form: $\boldsymbol{Q}(\boldsymbol{\theta}) = \mathrm{diag}(10^{\theta_1}, \ldots, 10^{\theta_{n_x}})$, and $\boldsymbol{R}(\boldsymbol{\theta}) = \mathrm{diag}(10^{\theta_{n_x+1}}, \ldots, 10^{\theta_{n_x+n_u}})$, such that the number of optimization parameters is reduced to $n_x + n_u$.

For the remainder of this paper, we refer to the aforementioned methods as K-learning [3], AB-learning [9], and QR-learning [8], respectively. In this paper, we propose to combine the ideas of K-learning and the LQR to allow for efficient optimization of high-dimensional policies. This is achieved by selecting the initial optimization domain boundaries based on a probabilistic dynamics model and the LQR. Additionally, the model-based approach allows us to find a linear embedding to reduce the effective dimensionality of the optimization problem as will be described in the next section.

IV. AUTOMATIC DOMAIN SELECTION & DIMENSIONALITY REDUCTION

It is well-known that BO has to cover the parameter space sufficiently well with respect to the lengthscale of the cost function in order to find a good estimate of the true optimum. Without prior knowledge, however, it is difficult to decide on the range of the parameters for optimization, which is crucial for obtaining good performance without spending an excessive amount of function evaluations for exploration. Commonly, the issue of finding an appropriate domain is left as tuning parameter and domains are chosen, e.g., by prior experience or problem-specific expertise. Especially in high dimensions, manual tuning of the domain parameters is not feasible.

We address this issue and propose a technique for automatic domain selection, which consists of the following steps: first, a probabilistic model for the system dynamics is learned and, second, we then employ model-based techniques from optimal control to find a distribution over policies. Based on this distribution we can define an appropriate domain for tuning the policy parameters using BO (Sec. IV-A). Furthermore, we can use the distribution over policies to find an embedding that maps the policy parameters into a lower-dimensional space, thus further increasing the efficiency of the subsequent optimization (Sec. IV-B).

To obtain a probabilistic model of the system dynamics, we perform Bayesian linear regression (see, e.g., [21, §3.3]) using recorded data of state/action trajectories to obtain an approximate linear model of the system dynamics. This results in a Gaussian distribution over linear dynamics models:

$$p(\texttt{vec}(\boldsymbol{A}, \boldsymbol{B})|Data) = \mathcal{N}(\texttt{vec}(\boldsymbol{A}, \boldsymbol{B})|\boldsymbol{\mu}^{AB}, \boldsymbol{\Sigma}^{AB}), \tag{5}$$

where $\boldsymbol{\mu}^{AB}$ is the maximum posterior (MAP) estimate, $\boldsymbol{\Sigma}^{AB}$ quantifies the distribution's uncertainty, and the $\texttt{vec}(\cdot, \cdot)$ notation denotes that the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are reshaped and stacked to a vector.

A. Independence Domain

Based on the probabilistic model of the system dynamics, we are now seeking to define an appropriate range for all policy parameters. From Eq. (5) we can sample n_s pairs of $(\boldsymbol{A}, \boldsymbol{B})$ for which we can each calculate the respective LQR feedback gain matrix, resulting in the sample distribution: $p(\texttt{vec}(\boldsymbol{K})|(\boldsymbol{A}, \boldsymbol{B})_{1:n_s})$. In general, this sample distribution can be multi-modal due to nonlinearities from the Riccati equation that is solved for the construction of the LQR [20]. However, since we are only looking for a bounding box based on the samples, it is sufficient to use a unimodal approximation. To this end, we use a product of independent normal distributions, one for each dimension:

$$p(\texttt{vec}(\boldsymbol{K})|(\boldsymbol{A}, \boldsymbol{B})_{1:n_s}) \approx \prod_{i=1}^{n_\theta} \mathcal{N}(\texttt{vec}(\boldsymbol{K})_i|\mu_i^K, \sigma_i^K), \tag{6}$$

where the individual parameters for means and variances are found using moment matching. Given this approximation, we can construct a domain centered around the mean of the distribution, where the width is governed by the distribution's standard deviation:

$$\begin{aligned}\mathbf{\Theta}^K_{\text{indep}} &= [\mu^K_1 - \beta\sigma^K_1, \mu^K_1 + \beta\sigma^K_1] \\ &\quad \times \cdots \times [\mu^K_{n_\theta} - \beta\sigma^K_{n_\theta}, \mu^K_{n_\theta} + \beta\sigma^K_{n_\theta}],\end{aligned} \tag{7}$$

where the parameter β determines the effective size of the domain. Due to the assumption of independence between entries in $\boldsymbol{K}$, we call this domain the *independence domain*.

B. PCA Domain

In the previous section, we assumed independence between policy parameters of the sample distribution. However, in general the entries of $\boldsymbol{K}$ are not independent. In order to take advantage of potential correlations between parameters, we can approximate the sample distribution with a multivariate Gaussian:

$$p(\texttt{vec}(\boldsymbol{K})|(\boldsymbol{A},\boldsymbol{B})_{1:n_s}) \approx \mathcal{N}(\texttt{vec}(\boldsymbol{K})|\boldsymbol{\mu}^K, \mathbf{\Sigma}^K), \tag{8}$$

where the goal of the approximation is to model the overall location and spread of the samples and not to accurately model the (potentially) multiple modes.

Now, based on the multivariate distribution, we propose to transform the optimization parameters into the eigenspace of the covariance matrix, $\mathbf{\Sigma}^K$. The transformation of the parameters is then described by $\tilde{\boldsymbol{\theta}} = \boldsymbol{T}(\boldsymbol{\theta} - \boldsymbol{\mu}^K)$, where the transformation matrix $\boldsymbol{T}$, consists of the eigenvectors of $\mathbf{\Sigma}^K$. In the eigenspace, the optimization domain is given by:

$$\tilde{\mathbf{\Theta}}^K_{\text{PCA}} = [-\beta\tilde{\sigma}^K_1, \beta\tilde{\sigma}^K_1] \times \cdots \times [-\beta\tilde{\sigma}^K_{n_\theta}, \beta\tilde{\sigma}^K_{n_\theta}], \tag{9}$$

where $\tilde{\sigma}^K_i$ denotes the i-th eigenvalue of $\mathbf{\Sigma}_K$. In essence, we are performing a principal component analysis (PCA) [21, §12.1] and hence we call this domain the *PCA domain*. The benefit of using this kind of transformation is that we 1) are able to identify the most relevant directions of the parameter space and 2) still retain the uncertainty information in each direction and thus are able to create meaningful parameter ranges in the transformed space.

For high-dimensional problems, some of the eigenvalues are often close to zero and the domain size in the respective dimension becomes negligible, effectively reducing the dimensionality of the optimization problem. Note that the volume of the domain in eigenspace is always smaller or equal to the independence domain, thus BO in the eigenspace leads to faster convergence if there are correlations between parameters. A visualization of the different domains is shown in Fig. 1.

C. Dynamic Domain Adaptation

For both the independence domain and PCA domain presented in the previous sections, the tuning parameter β needs to be chosen carefully. With increasing β, BO has to cover a larger space during the subsequent optimization, which means that more evaluations of the cost function are needed for sufficient exploration. At the same time, it

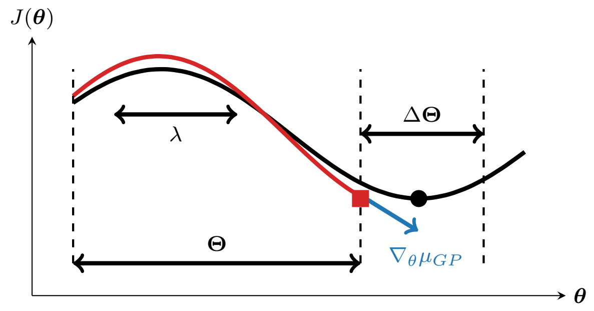


Fig. 2: Sketch depicting the relevant parameters used for DDA. The true objective (▬) is approximated by a GP (▬) and the estimated optimum (■) is on the domain's boundary. Consequently, the domain grows in the direction of the global optimum (●) with stepsize $\Delta\mathbf{\Theta}$ which is proportional to the GP's lengthscale λ, the GP's gradient $\nabla_\theta\mu_{GP}$ at the estimated optimum and the size of the current domain $\mathbf{\Theta}$.

might also be possible to find better policy parameters on a larger domain. Thus, the goal is to find a trade-off between restricting the domain to a reasonable size and choosing a large enough domain such that we do not suffer from model bias. We argue that it is more efficient to start with a small domain, e.g., choosing $\beta = 0.5$, and then adapting the domain boundaries during optimization to account for the fact that the global optimum might not lie within the initial domain. In this section, we explain how to exploit the surrogate model that approximates the objective function in order to adapt the domain boundaries in a goal-oriented manner.

While running BO, we have an estimate of the optimum, $\boldsymbol{\theta}^*$, i.e., the minimum of the approximate cost function on the current domain. If BO finds the location of the estimated optimum to lie on the domain's boundary, $\partial\mathbf{\Theta}$, it is likely that there are better parameters outside the current domain. Thus, we propose to grow the domain in the dimensions for which the estimated optimum is at the boundary.

This *dynamic domain adaptation (DDA)* is guided by the GP that models the objective function. The stepsize, $\Delta\mathbf{\Theta}_i$, by which the boundaries are increased is chosen heuristically proportional to three factors:

1) The gradient of the GP posterior mean at the current estimate of the optimum, $\nabla_{\theta_i}\mu_{GP}(\boldsymbol{\theta}^*)$. If the gradient at the boundary is steep, we expect a potentially better point to be further away from the boundary than if the gradient is small.
2) The lengthscale, λ_i, of the GP that approximates the cost function. For large lengthscales, the model assumes the cost function to vary slowly and thus the stepsize should be increased accordingly.
3) The domain's extent in dimension i. Dimensions in which the domain is large should also be increased by a greater stepsize.

The stepsize is then given by

$$\Delta\boldsymbol{\Theta}_i = \gamma \cdot \nabla_{\theta_i}\mu_{GP}(\boldsymbol{\theta}^*) \cdot \lambda_i \cdot \|\boldsymbol{\Theta}_i\| \,, \qquad (10)$$

where γ is a tuning parameter that governs the effective step size. A one-dimensional visualization of DDA and all its relevant parameters can be seen in Fig. 2 and a summary of the proposed algorithm is described in Alg. 1. Note that DDA is not limited to policy search but can be used for any BO procedure irrespective of the application and the meaning of the parameters.

A similar heuristic to grow the domain during optimization has been proposed in [22], where the volume of the domain is increased isotropically by a constant factor every few evaluations of the objective function. In the provided experiments, the constant factor is chosen to be 2, and thus this approach is called volume doubling (VD). Our approach differs in two aspects: 1) the growth of the domain is not based on a fixed schedule, i.e., increasing the domain every few iterations and 2) we increase the domain anisotropically based on the surrogate model. We argue that in the case of an initial domain that is already close the objective function's global optimum, DDA only increases the domain towards the global optimum and thus leads to faster convergence compared to VD. Besides the VD heuristic, another approach has been proposed in [22] that regularizes the acquisition function with a quadratic prior mean function. In this way, no explicit domain boundaries need to be specified, however the shape of the quadratic regularizer gives rise to implicit bounds. All of the aforementioned methods are evaluated on a synthetic 2D function in Sec. V-A.

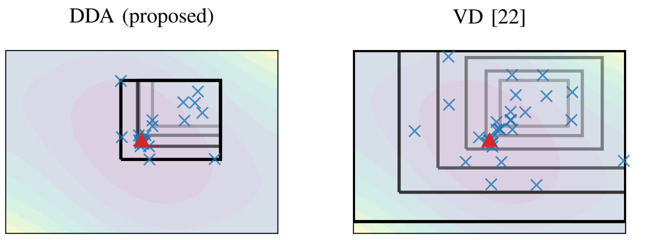
Algorithm 1 BO with DDA and domain selection

```
1: $p(\mathrm{vec}(\boldsymbol{A},\boldsymbol{B})|Data) \leftarrow$ perform system identification to
   obtain probabilistic dynamics model in Eq. (5)
2: $\boldsymbol{\Theta} \leftarrow$ select initial optimization domain, e.g., based on
   dynamics model (see Sec. IV)
3: **repeat**
4:     $\boldsymbol{\theta}^* \leftarrow$ run BO on current domain $\boldsymbol{\Theta}$
5:     **if** $\boldsymbol{\theta}^* \in \partial\boldsymbol{\Theta}$ **then**
6:         $i \leftarrow$ dimension at which $\boldsymbol{\theta}^*$ is at $\partial\boldsymbol{\Theta}$
7:         $\boldsymbol{\Theta}_i \leftarrow$ increase domain by stepsize $\Delta\boldsymbol{\Theta}_i$
           with $\Delta\boldsymbol{\Theta}_i \propto \nabla_{\theta_i}\mu_{GP}(\boldsymbol{\theta}^*)$, $\lambda_i$, $\|\boldsymbol{\Theta}_i\|$
8:     **end if**
9: **until** convergence or sufficient performance is achieved
```

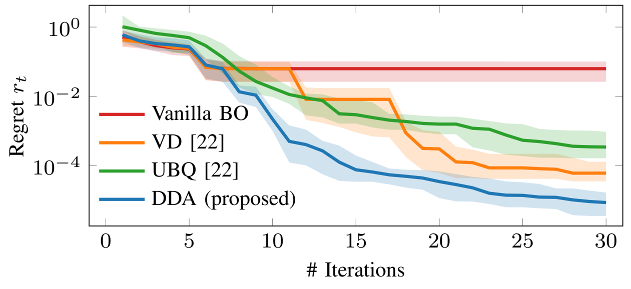
V. SIMULATIONS AND EXPERIMENTS

With the simulations and hardware experiments presented in this section, we aim at supporting the following results:

- The methods proposed in Sec. IV are able to select meaningful domain boundaries for BO and are applicable for a variety of control tasks with less manual parameter tuning.
- The proposed DDA scheme helps to adjust the domain boundaries in a goal-oriented manner and is more efficient than the VD scheme if the initial domain is already close to the objective function's global optimum.
- Choosing an informed parameter transformation reduces the number of required experiments needed for good performance significantly and enables BO based policy search for higher-dimensional systems.



(a) Parameter space view showing the growing domain boundaries (▬) and evaluated points (×) during optimization. The global optimum is marked by ▲.



(b) Regret of the best function value seen so far. With DDA the regret converges faster as the domain growth does not follow a fixed schedule (VD) and has more flexibility compared to UBQ.

Fig. 3: Comparison of dynamic domain adaptation (DDA), volume doubling (VD), unbounded quadratic (UBQ) and vanilla BO on the three-hump camel function.

A. Synthetic Function

We start by comparing the DDA scheme presented in Sec. IV-C with the approaches proposed in [22]. As objective to be minimized, we choose the three-hump camel function, a 2-dimensional function with three local and one global optimum at $\boldsymbol{\theta}^* = (0, 0)$. We sample $N = 50$ random initial domains of size 0.5 in each dimension, where it was ensured that the global optimum was not contained in the initial domains. We report the regret $r_t = |f(\boldsymbol{\theta}^*) - \min_t f(\boldsymbol{\theta}^t)|$ for all N initial domains and illustrate the domain adaptation for one of the sampled initial domains in Fig. 3. The solid line represents the median over all runs and the shaded areas indicate the 25th and 75th percentile, respectively. The example highlights that DDA leads to improved convergence and more goal-oriented sampling of the parameter space.

B. Policy Search

In this section we compare the different policy parameterizations presented in Sec. III and evaluate the influence of the proposed methods in Sec. IV and Sec. IV-C on K-learning. Additionally, we benchmark our method to REMBO [5], a well-known method to reduce the dimensionality of the

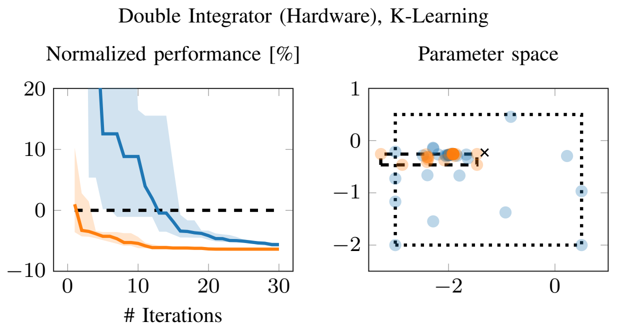


Fig. 5: Left: Comparison of performance when policy is optimized on the large domain (▬) and the independence domain (▬). 10 independent runs were performed. Right: Parameter space showing large domain (- -), independence domain (···), evaluated policies (●/●) and nominal LQR (×).

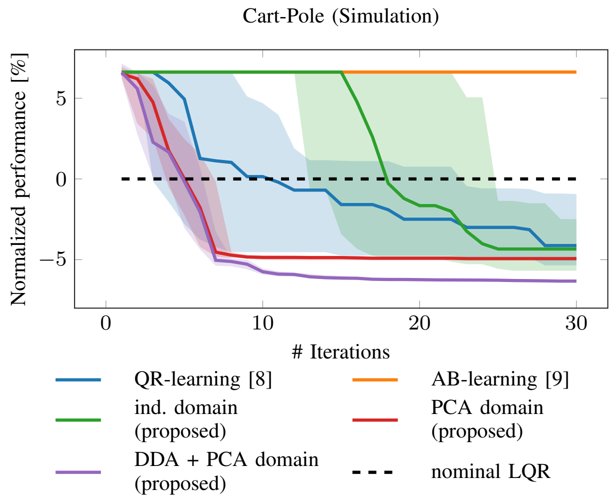


Fig. 6: Comparison of our proposed extensions to K-learning with QR- and AB-learning on the cart-pole task. The dashed line corresponds to the nominal LQR using the true (linearized around the upright position) dynamics. 30 independent runs were performed.

objective function using random embeddings. To evaluate performance of the policies, we compare the cost of one episode resulting from a learned policy to the cost resulting from the nominal LQR policy. In simulation, the nominal LQR policy is based on the true dynamics model linearized at the target position and for the hardware experiments, the nominal model provided by the manufacturer is used as true model. This results in the normalized performance measure $\eta = (J - J_{\text{LQR}})/J_{\text{LQR}}$. In the convergence plots shown in the following, the solid lines represent the median performance and the shaded areas indicate the 25th and 75th percentile, respectively. For all experiments we use the *GPyOpt*-Toolbox [23] and the UCB acquisition function [17], which was shown to outperform other acquisition functions in robotics applications [3]. In addition, we use the logarithm of the cost function (1), as this has been shown to lead to faster convergence [9].

For the hardware experiments we use the Quanser Qube Servo 2[3] which has two setups: one being a disk, which is a double integrator system, the second being a Furuta pendulum [24], see also Fig. 4.

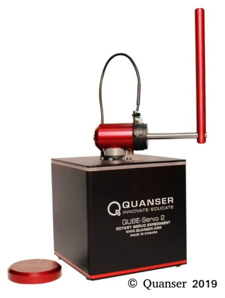


Fig. 4: Furuta pendulum used for hardware experiments.

a) Double Integrator (Hardware): The state of the double integrator is 2-dimensional and consists of the disk's angle, ϕ, and its velocity, $\dot{\phi}$. For system identification, we applied random inputs and recorded 5 seconds of data. Initially, the disk was placed at $\boldsymbol{x}_0 = [\phi_0, \dot{\phi}_0]^\intercal = [90°, 0]^\intercal$ with the goal of regulating it to the zero state.

In this experiment, we only apply K-learning, because this two-dimensional problem already shows the importance of domain selection and the domains can be easily visualized. For a comparison between AB-, QR- and K-learning, we consider the dimensionality of this problem too low and thus show it on more complex systems in the following sections.

Although the problem appears simple, vanilla BO on a large domain often fails to find a sensible policy (see Fig. 5). Optimizing on the large domain shows slow convergence as many evaluations are non-informative and far away from the optimum. When using the proposed method in Sec. IV to identify a relevant domain (independence domain in this case), BO consistently finds good policies in few iterations. The nominal policy performs sub-optimally due to effects from friction and stiction, which are not modeled.

b) Cart-Pole (Simulation): The cart-pole system consists of a cart that can be moved horizontally with a freely rotating pendulum. The input is a horizontal force acting on the cart. The state of the system is given by the cart's position, z, the pendulum's angle, ϕ, and their respective time derivatives: $\boldsymbol{x} = [z, \phi, \dot{z}, \dot{\phi}]^\intercal$. For system identification, we started with the pendulum in the upright position ($\phi = 0°$), applied random inputs and recorded the resulting state trajectory until the absolute value of the angle was larger than 30°. This process was repeated five times. The task was to stabilize the pendulum at the upright position over a time horizon of five seconds, where the initial condition was set to $\boldsymbol{x}_0 = [0, 45°, 2\frac{m}{s}, 0]^\intercal$. Note that with this initial condition, the system dynamics are strongly nonlinear.

In Fig. 6 we compare K-learning on the independence domain with AB-, QR-learning. Additionally, we show the improved performance of K-learning when we optimize on the PCA domain, with and without using DDA. QR-learning and K-learning on the independence domain show slow convergence and high variance in the resulting performance. AB-learning fails to improve within 30 iterations, which is in accordance to the results presented in [9], where several

[3]`https://www.quanser.com/products/qube-servo-2/`

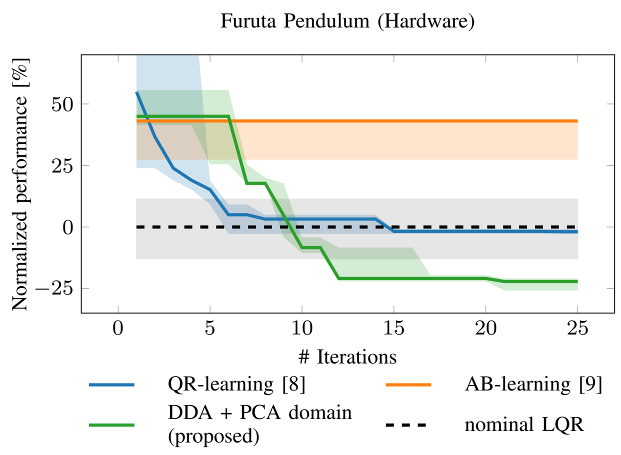


Fig. 7: Comparison of our proposed extensions to K-learning with QR- and AB-learning on the Furuta pendulum. 5 independent runs were performed.

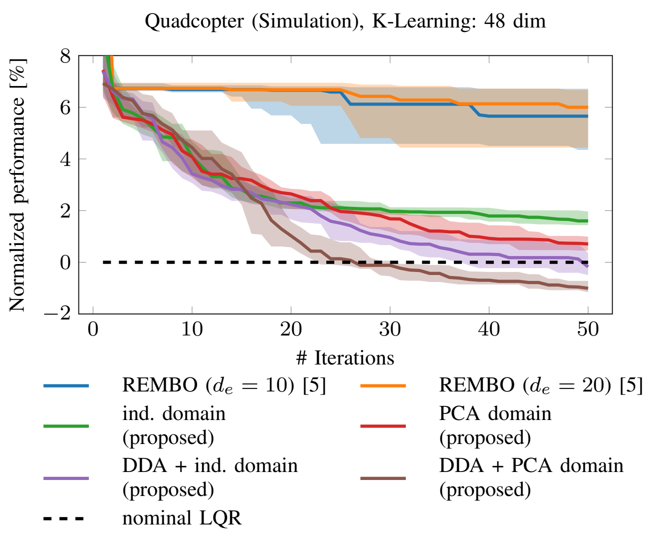


Fig. 8: Performance comparison obtained from directly optimizing the full, 48-dimensional feedback gain matrix of a quadcopter. Our proposed extensions are compared to REMBO with two different effective dimensionalities. 30 independent runs were performed.

hundred iterations are needed to achieve good performance for the same system and a task of similar complexity. Only when optimizing on the PCA domain using K-learning, we consistently outperform the LQR within only five iterations. However, as discussed in Sec. IV-C, it is possible to converge to a sub-optimal solution when the optimization domain is chosen too conservatively, as can be seen by the red curve that converges quickly to its final performance. When we additionally use DDA, the performance is further increased and all other methods are consistently outperformed.

c) Furuta Pendulum (Hardware): An inverted pendulum is attached to the end of a rotary arm that is actuated via a torque; the underlying system dynamics are similar to that of a cart-pole system. The state is given by $\boldsymbol{x} = [\eta, \phi, \dot{\eta}, \dot{\phi}]^\intercal$, with η being the angle of the rotary arm and ϕ the angle of the pendulum, respectively. The system identification process was the same as for the simulated cart-pole system and 5.7 seconds of trajectory data were used. The task to be optimized was regulating the system from $\boldsymbol{x}_0 = [45°, 0, 0, 0]^\intercal$ to the zero state, accumulating a cost over five seconds.

Qualitatively, the results on the hardware are similar to the simulation results on the cart-pole system (see Fig. 7)[4]. On the hardware, only K-learning with the proposed extensions consistently outperforms the nominal policy within $\sim$10 iterations. The performance of QR-learning stagnates quickly at the nominal controller's level, which might be due to the model-based policy parameterization. AB-learning is also unable to improve within the given number of iterations due to the high dimensionality of the optimization problem. The resulting policies during K- and QR-learning result in stable behavior of the pendulum in most cases. However, when using AB-learning on the hardware, we reject policies that are not in a pre-defined safe set, in order not to damage the experimental setup. For rejected policies, we assign the same cost as obtained from the initial policy.

d) Quadcopter (Simulation): To show the applicability to high-dimensional problems, we demonstrate the approach for a simulated quadcopter. This system has twelve states (position, linear velocities, orientation and angular velocities) and four inputs (rotational velocity of the rotors). For model learning, we recorded 50 seconds of flying through ten different waypoints which were uniformly sampled from $[x, y, z, \psi] \sim [-1m, 1m]^3 \times [-90°, 90°]$, with ψ being the yaw angle and the other states of the waypoints were set to zero. In hardware experiments, this procedure can easily be adapted by using a remote controller to generate the data. The control task was to stabilize the quadcopter at the hover position where the initial position was shifted $2m$ in the x-direction, and the roll and pitch angle were set to 30°.

Due to the superior performance of K-learning on previous experiments, we focus on this approach for the high-dimensional quadcopter and compare the influence of DDA and the different domains as introduced in Sec. IV. For the comparison with REMBO [5], we choose two different effective dimensionalities, $d_e = 10$ and $d_e = 20$, and optimize on the independence domain. For all methods, we initialize BO with the LQR policy obtained from the MAP estimate of the system identification step, which already was able to stabilize the quadcopter.

The results are shown in Fig. 8. With and without using DDA, optimizing on the PCA domain shows faster convergence in comparison with the independence domain due to

[4]All experiments were initialized with the same policy, i.e., the LQR using the MAP estimate. The spread of the performance can be explained by the inherent stochasticity of the problem on the one hand and small differences in the initial conditions that are inevitable for mechanical setups on the other hand. Also the nominal controller shows some variance in the resulting performance, where we use the median of 20 independent runs for cost normalization.

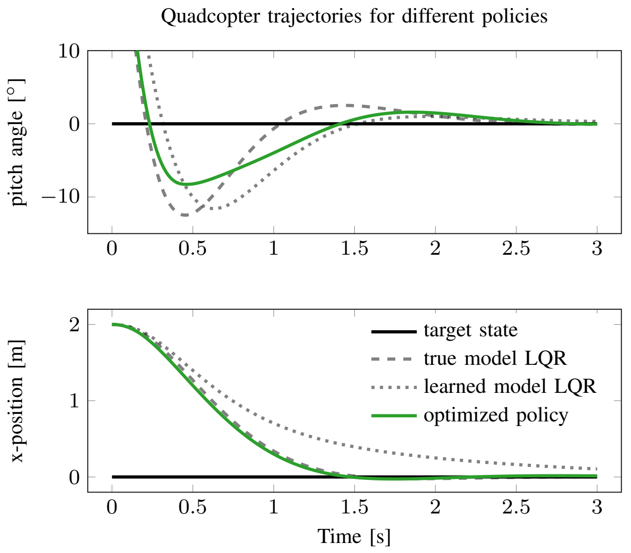


Fig. 9: Exemplary trajectories of the x-position and pitch angle before and after optimization. Note that the optimized policy leads to faster convergence to the desired states and less overshoot of the pitch angle.

the smaller volume of the domain. Furthermore, DDA helps to 1) further speed up the convergence and 2) allows for policies that consistently outperform the LQR within only 30 iterations. Using a random embedding for dimensionality reduction as done by REMBO shows significantly lower performance and none of the optimized policies is able to outperform the nominal LQR.

For a more intuitive understanding of the actual performance increase that is obtained with BO, we show trajectories of the quadcopter before (gray dotted) and after (green) optimization in Fig. 9. Especially the x-position of the quadcopter does converge significantly faster to the desired value after optimization of the policy.

VI. Conclusion

In this paper, we have shown that the choice of the optimization domain is critical for the convergence and final performance, as well as scalability of BO in policy search methods. By using a model-based technique from optimal control, we proposed an automatic domain selection method for optimizing a linear feedback policy in a model-free manner which exploits the objective functions structure and improves the sample efficiency of BO. Additionally, we introduce a dynamic domain adaptation mechanism in order to mitigate a potential model bias due to the choice of the initial domain. Simulations and experiments have shown that these contributions enable BO-based policy search techniques to find a policy that outperforms other control techniques that use the true dynamics model with only few system interactions. A key benefit of the reduced search domain provided by the proposed technique is the improved scalability of BO. We have demonstrated our approach to learn a 48-dimensional policy for a quadcopter—a size that renders standard BO techniques infeasible.

References

[1] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, 2012, pp. 2951–2959.

[2] J. Gonzalez, J. Longworth, D. James, and N. Lawrence, "Bayesian optimization for synthetic gene design," *NIPS Workshop on Bayesian Optimization in Academia and Industry*, 2014.

[3] R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth, "Bayesian optimization for learning gaits under uncertainty," *Annals of Mathematics and Artificial Intelligence*, vol. 76, no. 1-2, pp. 5–23, 2016.

[4] R. Martinez-Cantin, N. de Freitas, A. Doucet, and J. A. Castellanos, "Active policy learning for robot planning and exploration under uncertainty," in *Proceedings of the Robotics: Science and Systems Conference (RSS)*, 2007.

[5] Z. Wang, F. Hutter, M. Zoghi, D. Matheson, and N. de Feitas, "Bayesian optimization in a billion dimensions via random embeddings," *Journal of Artificial Intelligence Research*, vol. 55, pp. 361–387, 2016.

[6] R. Garnett, M. A. Osborne, and P. Hennig, "Active learning of linear embeddings for Gaussian processes," in *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2014, pp. 230–239.

[7] K. Kandasamy, J. Schneider, and B. Póczos, "High dimensional Bayesian optimisation and bandits via additive models," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2015, pp. 295–304.

[8] A. Marco, P. Hennig, J. Bohg, S. Schaal, and S. Trimpe, "Automatic LQR tuning based on Gaussian process global optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[9] S. Bansal, R. Calandra, T. Xiao, S. Levine, and C. J. Tomlin, "Goal-driven dynamics learning via Bayesian optimization," *arXiv preprint arXiv:1703.09260*, 2017.

[10] M. Frean and P. Boyle, "Using Gaussian processes to optimize expensive functions," in *Proceedings of the Australasian Joint Conference on Artificial Intelligence*, 2008, pp. 258–267.

[11] R. Antonova, A. Rai, and C. G. Atkeson, "Deep kernels for optimizing locomotion controllers," *arXiv preprint arXiv:1707.09062*, 2017.

[12] A. Wilson, A. Fern, and P. Tadepalli, "Using trajectory data to improve Bayesian optimization for reinforcement learning," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 253–282, 2014.

[13] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[14] H. J. Kushner, "A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise," *Journal of Basic Engineering*, vol. 86, no. 1, pp. 97–106, 1964.

[15] J. Močkus, "On Bayesian methods for seeking the extremum," in *Optimization Techniques IFIP Technical Conference*, 1975, pp. 400–404.

[16] D. D. Cox and S. John, "A statistical method for global optimization," in *IEEE Transactions on Systems, Man, and Cybernetics*, 1992, pp. 1242–1246.

[17] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2010.

[18] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[19] H. Mania, A. Guy, and B. Recht, "Simple random search provides a competitive approach to reinforcement learning," *arXiv preprint arXiv:1803.07055*, 2018.

[20] R. F. Stengel, *Optimal Control and Estimation*. Courier Corporation, 1986.

[21] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[22] B. Shahriari, A. Bouchard-Côté, and N. Freitas, "Unbounded Bayesian optimization via regularization," in *Artificial Intelligence and Statistics*, 2016, pp. 1168–1176.

[23] The GPyOpt authors, "GPyOpt: A Bayesian optimization framework in python," http://github.com/SheffieldML/GPyOpt, 2016.

[24] K. Furuta, M. Yamakita, and S. Kobayashi, "Swing-up control of inverted pendulum using pseudo-state feedback," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 206, no. 4, pp. 263–269, 1992.