Generalized Reinforcement Learning for Manipulation Skills – Combining Low-dimensional Bayesian Optimization with High-dimensional Motion Optimization

Peter Englert and Marc Toussaint

Abstract—This paper addresses the problem of how a robot can autonomously improve a manipulation skill in an efficient and secure manner. Instead of using the standard reinforcement learning formulation where all objectives are defined in a single reward function, we propose a generalized formulation that consists of three components: 1) A known analytic cost function; 2) A black-box reward function; 3) A black-box binary success constraint. While optimization of the analytic cost function is inherently high-dimensional, in typical robot manipulation problems we may assume that the black-box reward and constraint only depend on a lower dimensional projection of the policy. With our formulation we can exploit this structure and propose a sample-efficient learning framework that iteratively improves the skill with respect to the objective functions under the condition that the success constraint is fulfilled. The analytic cost function is optimized with motion optimization methods over the high dimensional policy where the lower dimensional parameters are fixed. The black-box reward is optimized with constraint Bayesian optimization over the low-dimensional parameter. During both improvement steps the success constraint is used to keep the optimization in a secure region and to clearly distinguish between motions that lead to success or failure. The learning algorithm is evaluated on simulated benchmark problems and real-world tasks like opening a door with a PR2.

I. INTRODUCTION

In this paper we address the problem of how a robot can improve its performance on a manipulation skill. In this problem, the essential question is how to select the next policy that leads to a performance improvement. This choice of selecting the next policy is strongly related to what kind of prior knowledge (e.g., objective functions) is available.

For many interesting tasks in robotics it is difficult to solve this problem because they consist of different subgoals and models are not always available. We would like to describe this difficulty with the example task of opening a door. Potential subgoals of this task are reaching the handle, pushing the handle down and doing the opening motion while keeping contact. Further goals are having a smooth motion and avoiding collisions. Combining all these subgoals into a single objective function and optimizing it is nontrivial. One reason for this is that all subgoals have to be weighted against each other and tuning these weights can be cumbersome. Another difficulty is the optimization of such an objective function since the properties of the cost terms differ (e.g., discrete/continuous, analytic/black-box). In our example the smoothness term is available analytically with gradients whereas the feedback if the door is open or not is only available as black-box function with a Boolean return.

A related issue is the availability of models. For the internal states of the robot a model usually can be obtained. However, for the environment (e.g., contacts, door joints) it is difficult to model it accurately.

In this paper we address these issues by introducing a generalized formulation of reinforcement learning that allows to express more structure in the problem definition. In this formulation the objectives are defined with three components: An analytic cost function, a black-box reward function and a black-box binary success constraint. The goal of this structuring is to separate the objective terms in such a way that it is possible to exploit knowledge (e.g., analytic form, models) when it is available and to explore it otherwise (i.e., black-box terms). For black-box functions it is difficult to optimize them efficiently in high-dimensional parameter spaces. Therefore, we make the assumption that the black-box reward and constraint only depend on a lower dimensional projection of the policy. In the area of manipulation skills this assumption is in many cases reasonable. One example is the manipulation skill of pushing a button where the contact parts are most important and the motion towards and away from the button are less crucial for success.

Using these three functions and the lower dimensional parametrization instead of the standard reinforcement learning setup has multiple benefits. One benefit is that there is less weighting required between the different objective terms since they are separated in different components. Another benefit is that the analytic cost function can be optimized efficiently over the full policy with efficient motion optimization methods. Whereas the black-box reward function can be optimized over the low dimensional parametrization that allow the efficient usage of global optimization methods. The advantage for using a binary success constraints is that it clearly allows to distinguish between policies that lead to task success or failure and reduces the number of failures with the system.

Later in the paper we use this problem formulation for the task of opening a door with a PR2. In this experiment we use as analytic function the smoothness of the motion, as black-box objective function we use the applied force and as success criteria we use the binary feedback if a policy opens a door. As lower dimensional parametrization we use the contact points on the door handle.
For this generalized reinforcement learning formulation we develop a learning algorithm that optimizes the objective functions under the condition that the success constraint is fulfilled. As a starting point we assume that the robot is provided with a successful policy of the task (e.g., by demonstration or hand-programmed). From this initialization the robot should improve the task with respect to the given objective functions and task success constraint. For achieving this goal we iterate between two improvement strategies. The first improvement strategy is Gauss-Newton motion optimization that optimizes the analytic cost function. Thereby we fix the lower dimensional parameter by incorporating them as constraints into the trajectory optimization method. The second improvement strategy is Bayesian optimization over the lower dimensional parameter. This method aims at maximizing the black-box reward function subject to the unknown constraint. We use Gaussian processes to learn models for the black-box objective function and constraint. These models are combined in an acquisition function to select the next policy in a secure and efficient manner. We introduce an algorithm that iterates between the analytic motion optimization and black-box Bayesian optimization strategies.

The main contributions of this paper are: 1) The generalized reinforcement learning formulation with an analytic cost function, a black-box reward function and a black-box success constraint. 2) A learning framework that improves the skill by combining motion optimization and Bayesian optimization. 3) A novel acquisition function for Bayesian optimization for selecting the next policy in a secure and efficient manner.

In the next section we present the generalized reinforcement learning formulation and an overview of our learning algorithm. Afterwards, we describe in Section III and IV the details of the motion optimization and Bayesian optimization. In Section V we discuss related work on reinforcement learning for manipulation skills and Bayesian optimization with constraints. In Section VI we evaluate our approach on synthetic and real-world problems.

II. GENERALIZED REINFORCEMENT LEARNING COMBINING ANALYTIC AND BLACK-BOX OBJECTIVES

A. Problem Formulation

We consider motor control policies that, in a given environment, generate a trajectory $\bar{x}(w)$ depending on policy parameters $w \in \mathbb{R}^n$. We specifically aim to deal with cases where $w$ is high-dimensional ($n \geq 1000$), e.g., a dense set of via points or spline parameters in joint or task space together with control parameters around this trajectory, or a large number of weights of basis functions that parameterize a dynamic movement primitive. But at the same time we aim for efficient skill learning from only few (< 100) real-world episodes, that is, evaluations of the different parameters $w$ by unrolling the policy and evaluating the task objectives. Clearly, for this to be a well-posed problem we need to assume additional structure in the problem.

We propose a generalization of the standard reinforcement learning (RL) formulation to capture such structure as follows. Given policy parameters $w$, the problem is defined in terms of a known analytic cost function

$$J(w) : \mathbb{R}^n \rightarrow \mathbb{R},$$

(1)

a $q$-dimensional equality constraint

$$h(w, \theta) = 0, \quad h : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^q$$

(2)

that ties every parameter $w$ to a lower-dimensional projection $\theta \in \mathbb{R}^m$ (see details below), a black-box reward function

$$f(\theta) : \mathbb{R}^m \rightarrow \mathbb{R},$$

(3)

and a black-box success constraint

$$s(\theta) : \mathbb{R}^m \rightarrow \{0, 1\}.$$

(4)

The generalized RL problem is

$$\min_{w, \theta} J(w) - f(\theta) \quad \text{s.t.} \quad h(w, \theta) = 0, \quad s(\theta) = 1.$$  

(5)

That is, we want the best policy $\bar{x}(w^*)$ (measured with $J(w)$ and $f(\theta)$) that fulfills a task (measured with $s(\theta) = 1$).

In addition to the standard reinforcement learning problem that only maximizes a single reward function $f(\theta)$, our formulation additionally consists on an analytic cost function $J(w)$, an equality constraint $h(w, \theta)$ and a black-box success constraint $s(\theta)$. The analytic cost function $J(\bar{x})$ contains all the costs we know a priori in analytic form, e.g., smoothness of $\bar{x}$. The black-box reward function $f(\theta)$ and success constraint $s(\theta)$ are a priori unknown and we can only observe noisy samples by doing rollouts for a given input. The above problem formulation uses the equality constraint $h(w, \theta) = 0$ to define the relation between the high-dimensional $w \in \mathbb{R}^n$ and the lower dimensional $\theta \in \mathbb{R}^m$. We assume that $h$ is smooth and that, for given $w$, $h(w, \theta) = 0$ identifies a unique $\Theta(w) = \theta$. In that sense, $\theta$ is a projection of $w$. We chose to formulate this projection in terms of an equality constraint so that, for given $\theta$, the remaining problem on $w$ is a standard constraint non-linear program (NLP).

B. Alternating NLP with Bayesian Optimization

As input to our method we assume that we have an initial policy $\bar{x}$ (e.g., from demonstration) that fulfills a task ($s(\theta) = 1$). We solve the problem in Equation (5) by iterating between two improvement strategies. The first improvement strategy is Gauss-Newton motion optimization and acts on the high-dimensional $w$ to improve the analytic cost function $J(w)$. Thereby the lower dimensional parameter $\theta$ is kept fixed. For doing this we use the equality constraint $h(w, \theta) = 0$. Fixing the lower dimensional parameter $\theta$ means that the resulting motions $\bar{x}$ fulfill a certain property (e.g., the endeffector position at a certain timepoint). We assume that the task success only depends on $\theta$, which implies that all trajectories $\bar{x}$ that fulfill the constraint for a fixed $\theta$ lead to the same task outcome. The second improvement strategy is Bayesian optimization over $\theta$ that aims at improving $f(\theta)$ and fulfilling the constraint $s(\theta)$. We propose a new acquisition function $a(\theta)$ that
is optimized in each iteration to select the next parameter. We define \( a(\theta) \) in such a way that it explores the parameter space in a secure and data efficient manner by finding a good tradeoff between making large steps that potentially lead to risky policies and small steps that would require many rollouts. For achieving this goal we learn \( s(\theta) \) with a binary classifier to find the boundary between policies that lead to success or failure. This classifier is used to keep the exploration around the feasible region and reduce the number of (negative) interactions with the system.

The resulting algorithm CORL that combines both improvement strategies in a sequential manner is described in Algorithm 1. In the following two sections first the motion optimization and afterwards the Bayesian optimization are described in detail.

### III. Motion Optimization for Fixed \( \theta \)

In the first part of Algorithm 1 the goal is to improve the trajectory \( \bar{x} \) directly while keeping the lower dimensional parameter \( \theta \) fixed. For doing this we use a Gauss-Newton motion optimization framework [1]. This method improves \( \bar{x} \) w.r.t. \( \theta \) by computing Newton steps; we assume that these steps can be translated to updates of the policy parameter \( w \). In our experiments, the mean trajectory is linear in the parameters, \( \bar{x} = Bw \), where \( B \) are RBFs or B-spline basis functions, leading to \( n = KQ \) parameters for \( K \) basis functions and a \( Q \)-dimensional robot configuration. In general, Newton steps can be pulled back using a linearization \( \bar{x}(w) \).

#### A. Background on Gauss-Newton Motion Optimization

The goal of trajectory optimization is to find a trajectory \( \bar{X} \), given an initial configuration \( x_0 \), that minimizes a certain objective function

\[
J(\bar{x}) = \sum_{t=1}^{T} \phi_t(\bar{x}_t, \gamma_t)^2
\]

This defines the objective as a weighted sum over all time steps where the costs are defined in form of squared features \( \phi \). Each cost term depends on a \( k \)-order tuple of consecutive states \( \bar{x}_t = (x_{t-k}, \ldots, x_t, \bar{x}_t) \), containing the current and \( k \) previous robot configurations [1]. In addition to the robot configuration state \( \bar{x}_t \) we use external parameters of the environment \( \gamma_t \) to contain information that are important for planning the motion (parameters of the environment’s configuration, e.g. object positions). In addition to the task costs we also consider inequality and equality constraints

\[
\forall_t \quad g_t(\bar{x}_t, \gamma_t) \leq 0, \quad h_t(\bar{x}_t, \gamma_t) = 0
\]

which are analogous to features \( \phi_t(\bar{x}_t, \gamma_t) \) and can refer to arbitrary task spaces. The resulting optimization problem is

\[
\bar{x}^* = \arg\min_{\bar{x}} J(\bar{x})
\]

s.t. \( g(\bar{x}, \gamma) \leq 0 \)

\( h(\bar{x}, \gamma) = 0 \)

where \( g \) and \( h \) are vector functions that contain all constraints. We incorporate the constraints with the augmented Lagrangian method and solve the resulting problem with Gauss-Newton optimization [2]. Thereby, we exploit the structure of the gradient and Hessian for efficient optimization (see [1] for more details).

We utilize this general framework for doing trust region policy improvement and for the optimization of the phase profile of the policy. Both steps are described in the following.

#### B. Trust-region Policy Improvement

In this step we optimize the policy parameters with respect to smoothness criteria. For doing this we define features that measure the transition costs. During this step we fix the lower dimensional parametrization \( \theta \). We do this by including the constraint \( h(w, \theta) \) as equality constraints (cf. Equation (7)) into the optimization problem. This leads to the behavior that some parts of the motion defined by the lower dimensional representation \( \theta \) are fixed (e.g. endeffector position at certain time step) during the optimization.

We could solve this trajectory optimization problem in Equation 8 until it converges to a fixed-point solution. However, this would lead to potential large steps between different trajectory candidates. This could be problematic since it is not guaranteed that the found trajectory leads to task success. Therefore, we limit the stepsize \( \Delta \bar{x} = |\bar{x}^{(i+1)} - \bar{x}^{(i)}|^2 \) of the trajectories between two iterations with the inequality constraint \( \Delta \bar{x} \leq \varepsilon_{\max} \). This trust-region constraint guarantees that the current trajectory is close to the previous trajectory. Note also the strong relation to relative entropy policy search (REPS) which limits the divergence between the trajectory.
distribution} before and after the update [3]. In our case, the resulting problem is
\[
J_{\text{ traj}}(\tilde{x}) = \sum_{t=0}^{T} \left( \frac{x_t - 2x_{t-1} + x_{t-2}}{\Delta^2} \right)^2
\]
s.t. \( c(\tilde{x}, \theta) = 0 \)
\( \Delta \tilde{x} < \epsilon_{\text{max}} \)

We select the next trajectory \( \tilde{x}^{(i+1)} \) by minimizing the problem defined in Equation (9).

C. Phase Profile Optimization

One drawback of the previous improvement step is that the constraint \( c(\tilde{x}, \theta) \) keeps some parts of the trajectory fixed that are not improved by the optimization step. For achieving smooth motions also in the constraint parts we additionally optimize the phase of the trajectory and keep the geometry fixed. For doing this we use a formulation based on splines to evaluate the trajectory \( \tilde{x} \) of duration \( T \) continuously between \( t \in [0, T] \). We want to optimize the phase profile \( p(t) : [0, T] \to [0, 1] \) of this trajectory w.r.t. to transition costs. We discretize \( p(t) \) in \( K \) points \( p_{1:K} = [p_1, p_2, \ldots, p_K] \) that we use as optimization variables with the boundary conditions \( p_1 = 0 \) and \( p_K = 1 \).

As smoothness term we use the squared accelerations that results in an overall cost
\[
J_{\text{phase}}(s) = \sum_{i=1}^{K} \left( \frac{(-2 \tilde{x}(p_i T) + \tilde{x}(p_{i-1} T) + \tilde{x}(p_{i+1} T))/\Delta^2}{2} \right)^2 \]
\[+ (-2p_i + p_{i-1} + p_{i+1})^2. \quad (10)\]

The second term is a cost term directly on the acceleration of the phase variable. The resulting phase profile \( p_{1:K}^* \) defines a new phase trajectory \( \tilde{x}^{(i+1)} \) that is applied on the real system. Similar to the previous improvement step we also limit the stepsize \( \Delta \).

There are different ways of how the trajectory improvement and phase improvement can be combined with each other. In our experimental evaluation later in the paper we first apply the trajectory improvement steps until convergence and afterwards apply the phase improvement steps. Alternatively, one could also iterate between the two steps. It is also possible to extend the motion optimization part with other improvement steps that fit in the Gauss-Newton framework.

IV. REINFORCEMENT LEARNING OVER \( \theta \) WITH UNKNOWN SUCCESS CONSTRAINTS

We introduce a reinforcement learning method to improve the policy with respect to the lower dimensional parameter \( \theta \). The goal of this improvement strategy is to optimize the black-box reward function \( f(\theta) \) under the success constraint \( s(\theta) \) so as to have a low amount of negative interactions with the system.

We use Bayesian optimization to learn a binary classifier for the success constraint \( s(\theta) \) and a regression model for the reward function \( f(\theta) \). We propose a new acquisition function \( a(\theta) \) that combines both models in such a way that the next policy is selected in a secure and data-efficient manner.

We compute the full policy \( \bar{x} \) for each sample \( \theta \) based on the solution of the previous motion optimization \( \bar{x}^* \) and the constraint \( c(\bar{x}^*, \theta) \).

We briefly introduce some background on Gaussian processes and Bayesian optimization before introducing our reinforcement learning strategy.

A. Background on Gaussian Processes

For both function approximations we use Gaussian processes (GP). The advantage of GPs is that they can express a broad range of different functions and that they provide probability distributions over predictions. A Gaussian Process defines a probability distribution over functions [4]. We will first handle the regression and afterwards the classification case. A GP is fully specified by a mean function \( \mu(\theta) \) and a covariance function \( k(\theta_1, \theta_2) \). In the regression case we have data of the form \( \{\theta_i, r_i\}_{i=1}^d \) with inputs \( \theta_i \in \mathbb{R}^m \) and outputs \( r_i \in \mathbb{R} \). Predictions for a test input \( \theta_t \) are given by mean and variance
\[
\mu(\theta_t) = m(\theta_t) + k(\theta_t) \Sigma^{-1} R
\]
\( \Sigma = \Sigma(\theta_1, \ldots, \theta_d) \)
with \( k(\theta) \), Gram matrix \( K \) with \( K_{ij} = k(\theta_i, \theta_j) \), and training inputs \( \Theta = [\theta_1, \ldots, \theta_d] \) with corresponding targets \( R = [r_1, \ldots, r_d] \). In the binary classification case the outputs are discrete labels \( s \in \{-1, 1\} \) and we have data of the form \( \{\theta_i, s_i\}_{i=1}^d \). Here we cannot directly use a GP to model the output. Therefore, the GP models a discriminative function \( g(\theta) \) which defines a class probability via the sigmoid function,
\[
p(s = 1 | \theta) = \sigma(g(\theta)). \quad (13)
\]
Since this likelihood is non-Gaussian the exact posterior over \( g \) is not a Gaussian process—one instead use a Laplace approximation [5] to approximate it. For more details regarding GP we refer to [4].

B. Background on Bayesian Optimization

Bayesian optimization [6] is a strategy to find the maximum of an objective function \( f(\theta) \) with \( \theta \in \mathbb{R}^n \), where the function \( f(\theta) \) is not known in closed-form expression and only noisy observations \( r \) of the function value can be made at sampled values \( \theta \). These samples are collected in a dataset \( \{\theta_i, r_i\}_{i=1}^d \) that is used to build a GP model of \( f \). The next sample point \( \theta_t+1 \) is chosen by maximizing an acquisition function \( a(\theta) \). There are many different ways to define this acquisition function [7]. One widely used acquisition function is the probability of improvement [8] which is defined as
\[
\Pi_f(\theta) = P( \theta \geq f(\theta^+) ) = \Theta \left( \frac{\mu(\theta) - f(\theta^+)}{\sqrt{\Sigma(\theta)}} \right)
\]
with \( \theta^+ = \text{arg max}_{\theta \in \Theta} f(\theta) \), where \( \Theta \) is the normal cumulative distribution function. We will make use of this probability of improvement in our acquisition function and extend it for an exploration in a safe manner.
C. Reinforcement Learning over $\theta$

We want to improve the skill by optimizing the constraint $\theta$ with respect to $f(\theta)$ and fulfilling the constraint $s(\theta)$. For doing this we collect data of the form $D = \{\theta^{(i)}, p^{(i)}, s^{(i)}\}_{i=1}^d$ where $\theta$ are the parameters, $r$ is the reward and $s$ is the task outcome. We use this data to make the decision where to select the next sample $\theta^{(d+1)}$. We use a GP $g_f$ for modeling the reward function $f(\theta)$ and a classifier $\sigma(g_s)$ with GP $g_s$ for modeling the success function $s(\theta)$. The regression GP contains only data points that are feasible and lead to task success. The classification GP describes the feasible region of all $\theta$ that lead to task success. This region is incrementally explored with the goal to find the the maximum $f(\theta)$ and leads to task success.

For both GPs we use a squared exponential kernel function

$$k(\theta_1, \theta_2) = \sigma_d^2 \exp \left(- \frac{1}{2} (\theta_1 - \theta_2)^T \Lambda^{-1} (\theta_1 - \theta_2) \right)$$

where $\Lambda = \text{diag}([l_1^2, l_2^2, \ldots, l_d^2])$ is a matrix with squared length scales and $\sigma_d$ is the signal standard deviation. In the regression model $g_f$ we use a constant prior mean function of 0. For the classification model $g_s$ we use a constant prior mean function $m(x) = c$ to incorporate knowledge that regions where no data points are available yet the unfeasible class is preferred. Therefore we select a constant $c$ smaller than 0. This allows to keep the exploration close to the region where data points are available. We use the regression GP $g_f$ and classification GP $g_s$ to define an acquisition function that we use to select the next sample $\theta^{(d+1)}$.

For selecting the next data point we introduce the acquisition function

$$a(\theta) = [g_s(\theta) \geq 0] P_{g_f}(\theta) + [g_s(\theta) = 0] \forall_{g_s}(\theta)$$

where $[\cdot]$ denotes the indicator function. The first term describes the probability of improvement (cf. Equation (14)) of $g_f$ in the inner region of the classifier $g_s$. The second term is the predictive variance of the GP classifier $g_s$ on the decision boundary. The first terms focuses on exploiting improvement inside the feasible region and the second term focuses on exploring safely on the decision boundary.

We maximize Equation (17) to find the next sample $\theta^{(d+1)}$. Afterwards, we compute the full policy representation $w$ with

$$\bar{x} = \arg\min_x ||x^* - x||^2 \quad \text{s.t.} \quad h(x, \theta^{(d+1)}) = 0$$

where $x^*$ is the solution of the previous motion optimization method (see Algorithm [1]). This trajectory $\bar{x}$ is executed on the system and the observed reward and task outcome are added to the dataset $D$. This procedure is repeated until convergence.

V. RELATED WORK

A. Bayesian Optimization with Constraints

There exist many approaches how constraints can be incorporated into Bayesian optimization [9]–[12]. The core concept of these approaches is to use the probability if a constraint is fulfilled $P[g(\theta)]$ and combine it with other acquisition functions (e.g., expected improvement). There exist multiple variations how to formulate the acquisition function depending on the problem requirements. In Gardner et al. [11] for example they assume that the objective can also be evaluated in the unsafe region. In Gelbart et al. [10] they consider the case of decoupled observation of objective and constraint.

The main difference to our work is that we additionally want to have a secure learning process. For obtaining this we also use the variance of the constraint to guide the exploration on the decision boundary.

B. Safe Exploration

Schreiter et al. [13] propose a safe exploration strategy for a similar problem to ours. They want to optimize a function in a safe manner where the feasible region is unknown. For doing this they assume to observe a safety measure when samples are close to the boundary. This information is integrated into a differential entropy exploration criteria to select next candidates. They also provide an upper bound for the probability of failure. This approach would most likely lead to fewer failures during the exploration, but also requires additional information about the decision boundary that might be difficult to define. We compare our approach to this strategy on a toy problem in Section VI-A.

Another approach for a safe exploration is proposed in Sui et al. [14]. The strategy SAFE_OPT uses the concept of reachability to categorize the search space in different sets for safe exploration and exploitation. The next data point is selected by sampling the most uncertain decision. They also provide a bound on the sample complexity and can guarantee safety with high probability.

Polo [15] introduce a safe reinforcement learning approach that improves demonstrated behavior in a risk-sensitive manner. The behavior is represented with case-based reasoning techniques. The safety criterion is defined with the distance to the nearest neighbor that is limited with a threshold. The exploration is done by adding Gaussian noise to the current optimal actions. Our approach relies on trajectories as policy representation, whereas their approach uses case-based reasoning techniques. This allows them to use multiple trajectories as demonstration that allows higher generalization abilities. However, our goal was not to reach high generalization but instead to improve a single motion as much as possible. Our structuring of the problem with a low dimensional parameter space allows to exploit prior knowledge and a more efficient learning on the real system.

C. Reinforcement Learning in Robotics

Reinforcement learning is a widely used approach for learning skills in robotics [16]. One approach was proposed by Kober et al. [17] that uses dynamic movement primitives as policy representation and the policy search method PoWeR to learn the shape and properties of the motion. Usually this approach is initialized from demonstration and afterwards policy search methods improve the skill with respect to a defined reward function. The difference to our approach is
that we perform learning on two policy parameterizations. This allows to use efficient Gauss-Newton optimization routines for the parts of a motion where an analytic cost function is available. For the lower dimensional parametrization we use Bayesian optimization.

An alternative to this approach was proposed by Theodorou et al. [18]. They present a method to learn force control policies for compliant manipulation. The policy is initialized with position control from imitation and afterwards this policy is augmented with a force/torque profile that is learned with the reinforcement learning method PI2 [18]. They use a single cost function that combines smoothness terms, force/torque terms and tracking errors to the demonstration. The weighting of the different terms in the cost function is nontrivial. We avoid this weighting by using a more structured problem definition with three different components.

D. Combined Learning Approaches

There has been previous work on how learning methods can be combined with each other to more powerful algorithms. In Rückert [19] a reinforcement learning algorithm for planning movement primitives is introduced that uses a two-layered learning formulation. In an outer loop the policy search method covariance matrix adaptation optimizes an extrinsic cost function that measures the task performance. This policy search is over parameters that are used in the inner loop to define a cost function for a trajectory optimization problem. This problem is used to compute trajectories that are fed back as input to the extrinsic cost function. This formulation is similar to our formulation since it also uses a black-box objective function and an analytic objective function. The main conceptual difference to our approach is that they directly couple the objective functions with each other in a hierarchical way and only optimize the extrinsic objective function. The intrinsic objective function is only used to perform rollouts. In our formulation we optimize both objective functions sequentially. Additionally, we use a safety constraint to guide the learning in a secure manner.

Kupcsik et al. [20] propose a policy search method that combines model-free reinforcement learning with learned forward models. They learn probabilistic forward models of the robot and the task that are used to generate artificial samples in simulation. These samples are combined with real world rollouts to update the policy. The relative entropy policy search method is used to maximize the reward and balance the exploration and experience loss by staying close to the observed data. They demonstrate their approach on multiple challenging robot experiments. One of the main differences to our approach is that we divide the problem in model-based motion optimization that improves the motion efficiently and reinforcement learning that improves the task by exploring a lower-dimension representation. A further difference is that they learn a model of the task that is used in internal simulations, whereas we directly learn a model that maps parameters to reward.

VI. Experiments

We evaluate our approach in two experiments. The first experiment is a toy function where we compare our acquisition function from Section [IV] to two alternatives. The second experiment is on a PR2 where we optimize the smoothness and interaction forces for the task of opening a door.

A. Bayesian Optimization Evaluation

In this experiment we evaluate our acquisition function in Equation [17] on a toy function. We compare our method to safe active learning (SAL) [13] and to Bayesian optimization with GP-UCB [7]. Note that SAL assumes to observe something like the distance to the feasibility boundary in critical (but feasible) regions, which our method and GP-UCB do not observe. As benchmark problem we use the same problem as in [13] with \( f(\theta) = 10\sin(\theta - 10)/(\theta - 10) \) and a feasible region of \(-5 \leq \theta \leq 11\). We limit the search space to the region \( \theta \in [-15, 15] \). The starting point is sampled uniformly between \(-0.5\) and \(0.5\). We assumed that \( f(\theta) \) can only be evaluated inside the feasible region. For the GP-UCB method the data points outside the feasible region received a constant negative reward of \(-4\). For the regression GP \( g_r \) we used the hyperparameter \( l = 3\), \( \sigma_f = 3.5 \) and \( \sigma = 0.25 \). For the classification GP \( g_s \), we set \( l = 1.75\), \( \sigma_f = 10 \) and a constant prior mean of \(-7\). The result are visualized in Figure [1]. We evaluated the learning performance with the best candidate found so far, the maximal distance of a sample to the
boundary and the amount of failed samples. The figures show the mean and standard deviation over 100 experiments.

The results show that GP-UCB converges fastest after 13 iterations. However, it also reaches the highest number of failures and the sample with highest distance away from the feasible region. The SAL method is as expected the safest method with at most 1 failure—but recall that it assumes it can observe the distance to the boundary in critical regions, which ours does not. SAL is also the slowest method with 23 iterations. Our method performs between these two. It exhibits a very low number of near-boundary failures even without observing critical distance, and a significantly higher convergence speed than SAL.

B. Opening a Door with a PR2

In this experiment we address the task of opening a door with a PR2. Such kind of manipulation tasks are of special interest since for the part of the motion where the robot is moving freely good models are available but for the part where the robot interacts with objects it is hard to obtain good models. This results from the fact that the environment is usually not completely known (e.g., position of objects, kinematic structure, physical entities) and that information about how this environment can be manipulated into a certain goal state is not available.

Therefore, we apply Algorithm 1 where we use the contacts as lower dimensional parametrization. The task setup is shown in Figure 2(a). Starting from a demonstration via kinesthetic teaching the robot improves the task as much as possible. As analytic objective function $J(\bar{x})$ we use the sum of squared acceleration in configuration space of the motion with the goal to reach a smooth motion. As black-box reward $f(\theta)$ we use the amount of force to open the door measured with a force torque sensor in the robot hand. The success criteria $c(\theta)$ is the Boolean function that tells if the door was opened successfully. For achieving an autonomous learning we used markers on the door to measure if it opened successful and added a simple motion that closes the door after each trial. This allowed the robot to perform the learning on its own without human intervention.

The trajectory $\bar{x}$ consists of 150 timepoints in a 7 dimensional configuration space that leads to 1057 parameter for the full policy. As lower dimensional parameter $\theta$ we define two parameters in the contact space of the door handle. The first parameter is the finger position on the handle relative to the demonstration. The second parameter is the finger opening widths. The parameters for the regression GP $g_f$ we used are $l = 0.0417$, $\sigma_f = 0.1682$ and $\sigma = 0.0120$. For the classification GP $g_s$ we set $l = 0.02$, $\sigma_f = 10$ and a constant prior mean of $-7$. As stopping criteria for Algorithm 1 we set $\epsilon_0 = 0.003$ and $\epsilon_1 = 10^{-5}$.

The results of the motion optimization are shown in Figure 3. This figure shows how the motion optimization method improves the squared acceleration of the trajectory with the trajectory optimization (see Section III-B) strategies. Starting with a demonstration first 10 iterations of the stepwise trajectory improvement and afterwards 6 iterations of the phase optimization are executed. Between iteration 16 and 17 and 17 the reinforcement learning with respect to $\theta$ was performed with 40 iterations. Afterwards another 4 iterations of trajectory optimization and 5 iterations of phase optimization are executed until the change of policies was below the threshold.

The results of the reinforcement learning in the lower dimensional parameter space $\theta$ are shown in Figure 2(b). We did 40 rollouts to explore the black-box reward function $f(\theta)$ and constraint $s(\theta)$. From these 40 rollouts 26 were successful and 14 were failures. The blue dots are successful rollouts, the red dots are failures and the green dot shows the best parameter. The red regions denotes the classifier region that distinguishes successful and failure policies.

We compared our proposed Bayesian optimization strategy
CORL to Covariance Matrix Adaptation (CMA). CMA has been used previously to learn robot skills [19], [21], [22]. We applied both methods on the same reward function \( f(\theta) \) over 100 iterations. The results are shown in the table in Figure 4.

We use as performance measure the highest reward achieved during the 100 iterations and the failure rate with the system. All values are reported as mean and standard deviation over four runs. It can be seen that CORL reaches a lower failure rate with a very low standard deviation. This results from the fact that the boundary is explored with our acquisition function (see Equation (17)). CMA also finds a slightly worse policy than CORL.

Both, our method and CMA operate on the low-dimensional parameterization \( \theta \) that we proposed with the CORL framework, exploiting the combination with the analytic motion optimization. We tried other approaches that do not rely on this low-dimensional projection and the combination with an analytic motion optimizer: We performed experiments with dynamic movement primitives and PoWeR similar to [17].

For this we parameterized the shape and goal of the DMP, leading to a 96 dimensional parameter space. However, we could not achieve a noticable learning performance after 150 iterations. We assume that the black-box reward function that combines the amount of forces with path smoothness is not informative enough for this large parameter space. This reinforces the motivation for our general approach of disecting rewards into high-dimensional analytical and lower dimensional black-box parts.

The demonstration and learning process are shown in a video in the supplementary material.

VII. CONCLUSION

In this paper we introduced an approach to learn manipulation with a generalized reinforcement learning formulation that combines high-dimensional analytic motion optimization and low-dimensional constrained Bayesian optimization techniques. We evaluated our algorithm on a benchmark problem and demonstrated the real-world applicability on a door opening task. The goal of this approach was to improve the performance on a skill in an autonomous fashion. Future work will combine this work with more general policy representations like [23] to achieve skills with high generalization abilities.

<table>
<thead>
<tr>
<th>Method</th>
<th>Highest reward</th>
<th>Failure rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORL</td>
<td>0.451 ± 0.032</td>
<td>0.157 ± 0.021</td>
</tr>
<tr>
<td>CMA</td>
<td>0.412 ± 0.017</td>
<td>0.190 ± 0.102</td>
</tr>
</tbody>
</table>

Fig. 4: Comparison between CMA and CORL

REFERENCES