Bayesian Optimization for Learning from Randomized Simulations

Bayes'sche Optimierung für das Lernen von Randomisierten Simulationen
Bachelor-Thesis von Christian Eilers aus Osnabrück
September 2019

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Bayesian Optimization for Learning from Randomized Simulations
Bayes'sche Optimierung für das Lernen von Randomisierten Simulationen

Vorgelegte Bachelor-Thesis von Christian Eilers aus Osnabrück

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Dr. Michael Gienger
3. Gutachten: M.Sc. Fabio Muratore

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Christian Eilers, die vorliegende Bachelor-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

In the submitted thesis the written copies and the electronic version are identical in content.

Datum / Date: Unterschrift / Signature:

_____ _____

Abstract

Physics simulators offer a powerful way to develop and test policies in robotics. Promising progress in learning difficult robotic control tasks has recently been made using reinforcement learning, however primarily in simulations due to the high sample complexity of state-of-the-art reinforcement learning algorithms. Transferring learned policies to the real world often fails because simulations only approximate the real world and are inherently imprecise. In recent years, domain randomization gained a lot of momentum for learning policies in simulation that are more likely transferable to the real world. However, the success of domain randomization depends on the design of the randomization distribution. In this thesis, we argue that the distribution can be adapted in order to maximize real-world performance of learned policies. We propose, how Bayesian optimization can be used to iteratively adapt the domain randomization distribution using reward signals of the real-world system. Finally, we present empirical results on a Furuta pendulum that the likelihood of learning transferable policies can be substantially increased by optimizing the domain parameter distribution.

Zusammenfassung

Physik Simulatoren bieten eine leistungsstarke Methode zum Entwickeln und Testen von Policies in der Robotik. Vielversprechender Fortschritt im Lernen von schwierigen Robotik Aufgaben wurde in der letzten Zeit durch Bestärkendes Lernen erreicht, jedoch hauptsächlich in Simulationen aufgrund der hohen Datenkomplexität von modernen Bestärkendes Lernen Algorithmen. Das Transferieren von Policies, die in Simulationen gelernt wurden, in die reale Welt schlägt oft fehl, weil Simulationen die reale Welt nur approximieren und von Natur aus unpräzise sind. In den letzten Jahren gewann Domain Randomisierung viel Momentum um Policies zu Lernen, die mit höherer Wahrscheinlichkeit auf das reale System übertragbar sind. Der Erfolg von Domain Randomisierung ist jedoch oft abhängig von dem Design der Randomisierungs-Distribution. In dieser Thesis schlagen wir vor, dass die Distribution mit dem Ziel der Performance Maximierung auf dem realen System hin angepasst werden kann. Wir schlagen vor, wie Bayes'sche Optimierung eingesetzt werden kann, um die Randomisierungs-Distribution, mithilfe von Signalen des realen Systems, iterativ zu adaptieren. Zuletzt präsentieren wir empirische Ergebnisse auf einem Furuta Pendel, dass die Wahrscheinlichkeit Policies zu lernen, die transferierbar sind, durch das Optimieren der Parameter Distribution wesentlich erhöht werden kann.

Contents

1 Introduction **2**

2 Related Work **3**

3 Foundations **4**
3.1 Reinforcement Learning 4
3.2 Randomized Simulations 4
3.3 Bayesian Optimization 5

4 Method **12**

5 Experiments **13**
5.1 Platform 13
5.2 Setup 15
5.3 Results 16

6 Conclusion **20**

Bibliography **21**

Figures and Tables

List of Figures

3.1 Gaussian process hyperparameter optimization using the marginal likelihood 10
3.2 Comparison of acquisition functions for a 1-dimensional function 11

5.1 Furuta pendulum benchmark platform 14
5.2 Comparison of the domain parameter distributions 17
5.3 Training progress of PPO for different models 19

List of Tables

5.1 Quanser Qube domain parameters and constants 14
5.2 Boundaries for the search space of Bayesian optimization 16
5.3 Sim-to-real performance comparison between baselines and our method 17

Abbreviations, Symbols and Operators

List of Abbreviations

Notation	Description
BFGS	Broyden–Fletcher–Goldfarb–Shanno
CEM	Cross Entropy Method
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
EI	Expected Improvement
i.i.d.	independently and identically distributed
PI	Probability of Improvement
PPO	Proximal Policy Optimization
UCB	Upper Confidence Bound

List of Symbols

Notation	Description
A	set of all actions
J	performance measure
S	set of all states
$\boldsymbol{\eta}$	hyperparameters of a Gaussian process
$\boldsymbol{\psi}$	hyperparameter of the domain randomization distribution
$\boldsymbol{\tau}$	trajectory
$\boldsymbol{\theta}$	policy parameters
$\boldsymbol{\xi}$	domain parameters
ℓ	length-scale of a Gaussian process covariance function
$\mathbb{R}$	set of real numbers
$\mathscr{D}$	dataset
$\mathscr{M}$	Markov decision process
$\mathscr{X}$	set of all inputs

π	policy
σ_f	signal variance of a Gaussian process covariance function
σ_n	noise variance of a Gaussian process

List of Operators

Notation	Description	Operator
R	reward function for a trajectory	$R(\boldsymbol{\tau})$
α	acquisition function	$\alpha(\bullet)$
$\mathbb{E}$	expected value of a random variable	$\mathbb{E}[\bullet]$
$\mathbb{N}$	normal distribution, fully specified by mean and variance	$\mathbb{N}(\mu, \sigma^2)$
exp	natural exponential function	$\exp(\bullet)$
log	logarithm to the base 10	$\log(\bullet)$
k	kernel/covariance function of a Gaussian Process	$k(\bullet)$
p	probability distribution of a random variable	$p(\bullet)$
r	reward function for a state-action pair	$r(\mathbf{s}, \mathbf{a})$

1 Introduction

In current research, physics simulators of real world systems are often utilized for learning robotic control tasks. Using simulators in robotics is generally more cost efficient, less time-consuming and safer than using real-world robots. These advantages make it very attractive to learn and test robotic controllers in simulation and later transfer them to the real world. Furthermore, simulators offer the possibility to collect a large and a highly diverse amount of data in short time spans compared to real-world systems making simulators particularly interesting for reinforcement learning.
In recent years, promising progress on learning difficult, high-dimensional, continuous control tasks was made using reinforcement learning algorithms [1, 2, 3]. However, most demonstrations have been done on simulated systems due to the high sample complexity of many state-of-the-art reinforcement learning algorithms, especially model-free policy search algorithms. Learning complex robotic control tasks with these algorithms on real robots is often too time consuming and expensive or even unfeasible. Moreover, reinforcement learning algorithms need to naturally explore their environment during learning which can be undesirable or even dangerous on real-world robots. Therefore, training of policies directly on real robots has only been successful in rather restricted domains [4]. In order to generally utilize the power of model-free reinforcement learning algorithms for robotics, we need to transfer policies that are learned in simulation to the real system with minimal requirements of real world data.
Unfortunately, simulations have the major drawback that they are only capable of approximating the real world and are inherently imprecise. Subsequently, the transfer of policies learned in simulation to the real system often fails as the policies are unable to generalize to the real world. This problem is often referred to as the reality gap [5]. Bridging the reality gap is of high interest, because simulations offer a powerful method to learn challenging control task e.g. by reinforcement learning algorithms. The simplest approach to make policies learned in simulation more likely applicable to the real world is to increase the accuracy of the simulator towards the real world. The most commonly used technique is system identification which fits the given model using data from the real world. While system identification typically works exceptionally well, showing impressive results, it can also be error-prone. Typical causes for system identification to fail on closing the reality gap are too simple models, unmodelable physical effects like wear-and-tear and non-rigidity as well as too imprecise sensors. A widely used technique, that is build up on system identification, is to perturb the simulator by adding independently and identically distributed (i.i.d.) noise to e.g. observations and actions [5]. Other methods are fine-tuning policies on the real system, that are pre-trained in simulation, learning probabilistic models or domain randomization. In the following, we introduce domain randomization, also called randomized simulation, which currently gains a lot of momentum in the research of robotic control and vision-based tasks for bridging the reality gap.
Domain randomization is a method that randomizes the domain parameters of the simulator. In the context of robotics, these are usually, but not exclusively, the physics parameters. The underlying hypothesis is that large variability in simulation results in policies that are more robust towards modeling errors and exploitation. This robustness should then make learned policies more likely to generalize to the real-world system and to be transferable. The general goal is to maximize the direct-transfer performance, which is the expected performance on the real-world system, without any direct training in the real-world. This is typically abbreviated and referred to as the sim-to-real performance. Successful applications of domain randomization are among other things made at manipulating a cube in a robotic hand [6], the locomotion of a quadruped robot [7] as well as in a robot pushing [8] and grasping task [9].
One of the main challenges of domain randomization is the design of the domain parameter distribution. Currently, the distribution often relies on prior belief, in particular domain knowledge [10, 11]. However, a good source distribution is not always known. If a learned policy fails on the real-world system, adaption of the domain parameter distribution can be a tedious process. It is therefore desirable to automate the process of finding a good domain parameter distribution. In this thesis, we present a data-driven, automated process to find a good domain parameter distribution. Concrete, we investigate how Bayesian optimization can be used to adapt the distribution using the rewards received from the real-world system.

2 Related Work

The discrepancy between simulations and real systems, which leads to policies not being successful in the real world is called the reality gap. The problem of closing the reality gap has gained momentum in research within the past years. Classical work on bridging the reality gap is commonly based on system identification [12]. However, system identification is not a general solution for bridging the reality gap, especially if the model is too simple. In the context of reinforcement learning, model-based methods are used to learn real-world tasks. In general, model-based reinforcement learning algorithms use real-world sampled trajectories to learn a model, which can then be used in simulation to learn policies. Impressive results in terms of data-efficiency have been demonstrated, e.g. by Deisenroth et al. [13] through iteratively learning a probabilistic model using available real-world data and improving the policy based on the learned model. However, the quality of the learned model acts as bottleneck for the quality of the learned policy. Learning a good model is not a generally easy task and especially falls off for higher dimensional systems.

A different approach for transferring policies from simulation to reality is domain randomization. In recent applications, domain randomization led to promising results in zero-shot sim-to-real transfer tasks [14, 8, 6, 9, 7, 15]. Lowrey et al. [16] compare the performance difference between policies that are learned on a model from proper system identification, a model with bad system identification and under randomized simulation on a composition of robots that aim to push an object to a target location. While proper system identification showed the best performance on the real system, policies learned under randomized simulation also solved the task with slightly worse performance. The improper system identification led to policies that could not solve the task.

The domain parameter distribution is usually designed based on prior belief and adapted through trial-and-error when learned policies fail to transfer. Recently, multiple approaches have been presented that propose how to design the domain parameter distribution. Ramos et al. [17] extend the system identification method to be probabilistic by computing a full posterior over the simulation parameters. The posterior is then used as domain parameter distribution to learn policies.

The closest related methods to our approach are [10, 18, 19], which also use real-world data to iteratively adapt the domain parameter distribution after each learned policy that is evaluated on the real-world system. The method proposed by Chebotar et al. [10] iteratively adapts the distribution hyperparameters by minimizing the discrepancy between trajectories from the real-world system and simulation. Rajeswaran et al. [18] propose an algorithm to train robust, risk-aware policies in randomized simulation and combine their learning algorithm with a Bayesian update rule for the domain parameter distribution using trajectories collected from the real system. The update rule is in the fashion of Bayesian model-learning update rules. The idea, that the reward on the target system can optimized directly by changing the domain parameter distribution was also recently presented by Vuong et al. [19]. Instead of Bayesian optimization, the Cross Entropy Method (CEM) is used to update the domain parameter distribution and empirical results are only present for the transfer of policies between different physics simulators and not to real-world systems.

For scenarios where prior access to the real system is not possible or too costly, Mozifian et al. [11] explore the use of gradient-based optimization techniques to find domain randomization distributions without real world samples. Their optimization objective is defined as a trade-off between performance of the policy and entropy of the distribution. The goal is to solve the given task successfully on the most diverse set of scenarios possibly. When training on a wide range of scenarios, learning policies can suffer from high variance in the sampled trajectories. This is a problem Mehta et al. [20] aim to solve by learning a strategy for sampling the domain parameters subsequently to learning the policy. The strategy is designed to prefer environment instances that lead to better generalization in the policy by maximizing utility.

The goal of adapting the distribution during learning has also been subject to the work of Paul et al. [21], however with the goal of being robust to significant rare events and not for sim-to-real performance. Their approach is based on Bayesian optimization to sequentially optimize the distribution after each policy update with respect to maximizing the reward of the next policy. As Bayesian optimization is used during learning, the policy parameters have a major influence on the expected return and cannot simply be ignored like in our method. To model the relation between the union of policy parameters and domain parameter distribution hyperparameters to the expected reward, they use a fingerprint of the policy. This fingerprint then allows to use policies with high-dimensional parameter space like deep neural networks.

Finally, we want to highlight that system identification, model learning and domain randomization are not the only options for sim-to-real transfer. Recently, Yu et al. [22] presented to learn a family of policies that map state observations and physics parameters to an action by defining both as inputs to a neural network. After training a policy in simulation, Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is used to find the inputs to the neural network that result in the best return on the real system.

3 Foundations

In this chapter, we briefly introduce reinforcement learning and randomized simulation, and explain the foundations of Bayesian optimization.

3.1 Reinforcement Learning

A finite-horizon Markov decision process $\mathscr{M} = (S,\ A,\ P,\ R,\ p_0,\ \gamma,\ T)$ is characterized by the state space S, action space A, the probabilistic system dynamics $P : S \times A \times S \to \mathbb{R}_+$, a reward function $R : S \times A \to \mathbb{R}$, an initial state distribution $p_0 : S \to \mathbb{R}_+$, a reward discount factor γ and a fixed horizon T.
The discounted reward of a trajectory $\boldsymbol{\tau} = (\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{s}_T, \mathbf{a}_T)$ is defined as

$$R(\boldsymbol{\tau}) = \sum_{t=0}^{T} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t).$$

The goal of reinforcement learning is to find a policy $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})$ parameterized by $\boldsymbol{\theta}$ that maximizes the expected discounted reward over trajectories induced by the policy

$$\max_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\tau}\sim\pi_{\boldsymbol{\theta}}}[R(\boldsymbol{\tau})], \tag{3.1}$$

where $\mathbf{s}_0 \sim p_0, \mathbf{s}_{t+1} \sim P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and $\mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)$, see [23].
In the later described experiments we use the Proximal Policy Optimization (PPO) algorithm, introduced by Schulman et al. [3], due to its simple implementation and good empirical results across a wide range of scenarios.
PPO is a model-free reinforcement learning algorithm that uses policy gradients and actor-critic style for updating the policy parameters. We use the clipped objective function

$$\mathrm{L}^{CLIP}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t\Big[\min\big(r_t(\boldsymbol{\theta})\,\hat{A}_t,\ \mathrm{clip}(r_t(\boldsymbol{\theta}), 1-\varepsilon, 1+\varepsilon)\,\hat{A}_t\big)\Big],$$

where r_t is the probability ratio

$$r_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}_t|\mathbf{s}_t)} \tag{3.2}$$

and $\hat{A}$ is the advantage function estimate.

3.2 Randomized Simulations

To receive probabilistic system dynamics in simulation, we can define a distribution over the domain parameters $\boldsymbol{\xi} \sim p_{\boldsymbol{\psi}}(\boldsymbol{\xi})$, resulting in the probabilistic system dynamics $P_{\boldsymbol{\xi}\sim p_{\boldsymbol{\psi}}} = P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \boldsymbol{\xi})$. The domain parameter distribution is parametrized by $\boldsymbol{\psi}$, e.g. the mean and covariance of a normal distribution. This approach is called domain randomization or randomized simulation and extends the formulation of (3.1) to

$$\max_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \max_{\boldsymbol{\theta}} \mathbb{E}_{P_{\boldsymbol{\xi}\sim p_{\boldsymbol{\psi}}}}[\mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R(\boldsymbol{\tau})]]. \tag{3.3}$$

Instead of maximizing the expected discounted reward induced by the policy for a specific model $\mathscr{M}$, we now maximize over a parameterized model $\mathscr{M}_{\boldsymbol{\psi}}$.
The general scheme of learning a policy under randomized simulation is presented in Algorithm 1.

Algorithm 1 Learning from Randomized Simulation

Choose a *PolicyOptimizer* (e.g. PPO)
Initialize parameters of the policy $\boldsymbol{\theta}_0$
Specify the hyperparameters of the domain parameter distribution $\boldsymbol{\psi}$
for $n = 0$ to N **do**
    Sample domain parameters $\boldsymbol{\xi}_{0:k} \sim p_{\boldsymbol{\psi}}(\boldsymbol{\xi})$
    Collect trajectories $\boldsymbol{\tau}_{0:k}$ using policy $\pi_{\boldsymbol{\theta}_n}$ for $\boldsymbol{\xi}_{0:k}$
    $\boldsymbol{\theta}_{n+1} = PolicyOptimizer(\boldsymbol{\theta}_n, \boldsymbol{\tau}_{0:k})$
end for
return Policy $\pi_{\boldsymbol{\theta}_N}$

3.3 Bayesian Optimization

Bayesian optimization [24, 25, 26, 27] is a sequential, global optimization method concerned with finding the maximum of a function $f(\mathbf{x})$ over a compact set $\mathcal{X} \subset \mathbb{R}^d$

$$\max_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}). \tag{3.4}$$

The function $f(\mathbf{x})$ is typically a black box function, which means that there is no closed-form or derivatives available and evaluation is restricted to sampling. Further, observations y of the function f are assumed to be noise corrupted such that $\mathbb{E}[y|f(\mathbf{x})] = f(\mathbf{x})$. Bayesian optimization is particularly effective if the function f is costly to evaluate as it maintains a probabilistic model of the function and can utilize prior belief about the problem.
Given some prior knowledge and data of the form $\mathcal{D}_{1:n} = (\mathbf{x}_i, y_i) \mid i = 1, \ldots, n\}$, we can employ a probabilistic model in form of a Gaussian process, called the surrogate function. The surrogate function captures the expectation of what the objective function f looks like. The next sample location $\mathbf{x}_{n+1}$ is then determined by a utility that specifies how good it is to sample at a specific location. This function is called acquisition function $\alpha(\mathbf{x})$. The acquisition function trades off exploration and exploitation by taking the expectation and uncertainty (mean and covariance for a Gaussian process) of the surrogate function into account. Exploration favors location where the surrogate function is very uncertain while exploitation favors locations where the surrogate function expects high function values.
At each step of Bayesian optimization, the objective function f is evaluated at the maximizer of the acquisition function. Then, the surrogate function is updated by taking the new sample into account. A sketch of Bayesian optimization is presented in Algorithm 2.

Algorithm 2 Bayesian optimization algorithm

Specify the compact set $\mathcal{X}$
Initialize the dataset $\mathcal{D}_0$ by observing f at n_0 samples
Initialize and fit the surrogate model $\mathcal{M}$
for $n = 0, 1, 2, \ldots,$ **do**
    Select $\mathbf{x}_{n+1}$ by optimizing the acquisition function α : $\mathbf{x}_{n+1} \leftarrow \arg\max_{\mathbf{x} \in \mathcal{X}} \alpha(\mathbf{x}|\mathcal{M}, \mathcal{D}_n)$
    Sample $f(\mathbf{x}_{n+1})$
    $\mathcal{D}_{n+1} = \{\mathcal{D}_n, (\mathbf{x}_{n+1}, f(\mathbf{x}_{n+1}))\}$
    Update the surrogate model $\mathcal{M}$ using $\mathcal{D}_{n+1}$
end for
return Maximizer $\mathbf{x}^*$ of the surrogate model $\mathcal{M}$

3.3.1 Gaussian Process Regression

A Gaussian process can be thought of as an extension to the multivariate Gaussian distribution by placing a distribution over a function instead of a random variable. While a Gaussian distribution is fully specified by a mean and covariance, a Gaussian process is fully specified by its mean function $m : \mathcal{X} \to \mathbb{R}$ and covariance function $k : \mathcal{X} \to \mathcal{X} \times \mathcal{X}$. The covariance function is also called kernel function [28].
Gaussian processes are non-parametric models and can be used to solve regression problems. A regression problem is defined as finding a function $f(\mathbf{x})$ that maps inputs to continuous outputs based on the available data $\{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, n\}$, where $\mathbf{x} \in \mathbb{R}^d$ are the inputs and $y = f(\mathbf{x}) + \varepsilon$ the observations with i.i.d. Gaussian noise $\varepsilon \in \mathcal{N}(0, \sigma_n^2)$.
By placing a Gaussian process prior on the function f, such that

$$p(f(\mathbf{x})) = GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

with

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})],$$
$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x})(f(\mathbf{x}') - m(\mathbf{x}')],$$

the likelihood is then given as

$$p(y|f, \mathbf{x}) = \mathcal{N}(y|f(\mathbf{x}), \sigma_n^2).$$

Given data in the form of inputs $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ and corresponding noisy outputs $\mathbf{y} = [y_1, \ldots, y_n]$, we can predict the posterior probability distribution of the function value f_* for a given test point $\mathbf{x}_*$

$$p(f_*|\mathbf{X}, \mathbf{y}, \mathbf{x}_*) = \mathcal{N}(f_*|\mu_*, \sigma_*^2)$$
$$\mu_* = m(\mathbf{x}_*) + k(\mathbf{x}_*, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I})^{-1}(\mathbf{y} - m(\mathbf{X})) \tag{3.5}$$
$$\sigma_*^2 = k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I})^{-1} k(\mathbf{X}, \mathbf{x}_*). \tag{3.6}$$

While (3.5) and (3.6) define the point-wise predicted distribution for a given test input, they remain the same if $\mathbf{x}_*$ denotes multiple test inputs and are therefore the mean function and covariance function of the posterior Gaussian process. However, to make predictions with a Gaussian process given some data we first need to define the prior mean and covariance which let us incorporate prior knowledge and assumptions about how the underlying function looks like.

Mean

The most common choice for the prior mean is the zero function $m(\mathbf{x}) = 0$ and is also our choice in the later presented experiments. Another typical choice is a constant function which then yields an additional hyperparameter that can be optimized. Other choices like explicit basis functions are presented in detail by Rasmussen and Williams [28].

Covariance

The covariance function is typically the key component of a Gaussian process and defines most of the properties we expect from the underlying function. In the following, we introduce the commonly used squared exponential kernel k_{SE} as well as the Matérn kernel k_{M52} [28, 26]. The index 52 indicates that the hyperparameter for smoothing has a value of 5/2. The kernels with signal variance σ_f (scaling factor) and dimension specific length-scales ℓ_d, also called automatic relevance detection, are given as

$$k_{\text{SE}}(\mathbf{x}, \mathbf{x}') = \sigma_f \exp(-\frac{1}{2} r^2(\mathbf{x}, \mathbf{x}')),$$
$$k_{\text{M52}}(\mathbf{x}, \mathbf{x}') = \sigma_f (1 + \sqrt{5r^2(\mathbf{x}, \mathbf{x}')} + \frac{5}{3} r^2(\mathbf{x}, \mathbf{x}')) \exp(-\sqrt{5r^2(\mathbf{x}, \mathbf{x}')}), \tag{3.7}$$
$$r^2(\mathbf{x}, \mathbf{x}') = \sum_{d=1}^{D} \frac{(\mathbf{x}_d - \mathbf{x}'_d)^2}{\ell_d^2}.$$

The Matérn kernel k_{M52} is often more attractive for modeling processes of real world applications because the squared exponential kernel k_{SE} can be too smooth.
Although the Gaussian process is a non-parametric model, we still have hyperparameters $\boldsymbol{\eta} = (\ell_d, \sigma_f, \sigma_n)$ which heavily influence posterior predictions.

Marginal Likelihood

An estimate of the hyperparameters is typically computed by optimizing the marginal likelihood

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\eta}) = \int p(\mathbf{y}|f, \mathbf{X}) p(f|\mathbf{X}) df.$$

For Gaussian process regression with Gaussian noise the marginal likelihood can be expressed in analytic form as

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\eta}) = -\frac{1}{2}(\mathbf{y} - m(\mathbf{X}))^T (k(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I})^{-1} (\mathbf{y} - m(\mathbf{X})) - \frac{1}{2} \log |k(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}| - \frac{n}{2} \log(2\pi). \tag{3.8}$$

By differentiating this equation with respect to the hyperparameters $\boldsymbol{\eta}$ we have an optimization problem that can be solved by first order optimization algorithms [28]. However, especially when data is scarce, optimizing the marginal likelihood can lead to bad local optima as the data can be interpreted in multiple ways [27]. Figure 3.1 illustrates this

problem by visualizing the surface of the marginal likelihood for different sample seeds and different sample sizes for the 1-dimensional toy problem

$$f_{\text{toy}}(x) = \sin(x) - x^2 + 0.7x + 0.1\varepsilon, \qquad \varepsilon \sim \mathcal{N}(0,1). \tag{3.9}$$

When the marginal likelihood has no clear optima but rather a large plateau, so called over- and under-fitting can occur. Over-fitting is here interpreted as kernels that have a low length-scale and low noise variance, causing the model to interpolate the data. We refer to under-fitting, when the kernel has a high length-scale and high noise variance, treating every data point as noise from the prior mean.
To avoid over- and under-fitting, it is important to have strong priors for the hyperparameter optimization. For normalized inputs and standardized outputs good prior distribution for hyperparameters exist that work well for a wide range of scenarios. This can remove the dependency on the human to specify hyperparameter priors for each new problem setting. Another approach is to integrate the hyperparameters out to have a full Bayesian treatment that automatically weight the predictions from different interpretation of the data. In practice however, this has to be approximated by sampling which scales very bad with the dimension of the input data and is very sensible to the sampling density [26]. In our later presented experiments, we use point estimates of the hyperparameters by optimizing the marginal likelihood with strong priors and avoid undesirable predictions as in Figure 3.1.
After placing a prior on our unknown function and updating it to a posterior by considering the available evidence in form of data, we are left with determining a criterion of which point to evaluate next.

3.3.2 Acquisition Functions

The acquisition function, denoted by $\alpha : \mathcal{X} \to \mathbb{R}$, is concerned with selecting the next query point given a probabilistic surrogate model and thereby guides the search for the optimum. It is designed to trade-off exploitation of areas with high predicted function values and exploration of uncertain areas in its search space. By maximizing the acquisition function we can determine the most promising sample location

$$\mathbf{x}_{\text{next}} = \underset{\mathbf{x}}{\operatorname{argmax}}\ \alpha(\mathbf{x}),$$

where promising implies potentially high function values. Contrary to the unknown function f, function samples from the acquisition function are very cheap. Because of that, we can use global optimizers like CMA-ES or first-order optimization algorithms like the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm in conjunction with multi-starts to find the maximizer of the acquisition function.
Several acquisition functions have been proposed. In the following, we introduce some of the most common acquisition functions in the context of Bayesian optimization.

Probability of Improvement

The Probability of Improvement (PI) criterion was first introduced by Kushner et al. [29]. For models that yield a Gaussian posterior distribution like Gaussian processes, it is defined as

$$\alpha_{\text{PI}}(\mathbf{x}) = \Phi\left(\frac{\mu(\mathbf{x}) - f_{\max}}{\sigma(\mathbf{x})}\right),$$

where $\Phi(\cdot)$ is the standard normal cumulative distribution function and $f_{\max}$ is the best current function value.

Expected Improvement

The Expected Improvement (EI) criterion can be interpreted as an extension to the PI criterion. For a Gaussian process and other models that have a Gaussian posterior, it can be expressed in closed form

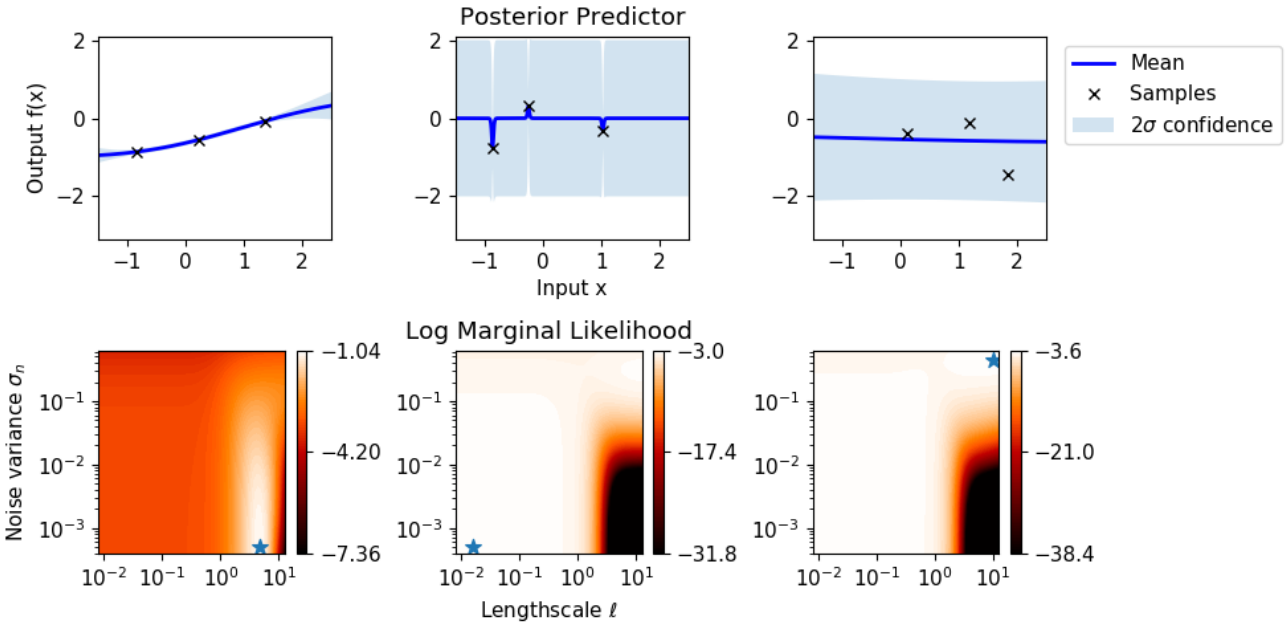
$$\alpha_{\text{EI}}(\mathbf{x}|\mathscr{D}) = (\mu(\mathbf{x}) - f_{\max})\Phi\left(\frac{\mu(\mathbf{x}) - f_{\max}}{\sigma(\mathbf{x})}\right) + \sigma(\mathbf{x})\phi\left(\frac{\mu(\mathbf{x}) - f_{\max}}{\sigma(\mathbf{x})}\right), \tag{3.10}$$

where $\phi(\cdot)$ is the standard normal probability density function. It was introduced in the context of Bayesian optimization by Jones et al. [24].

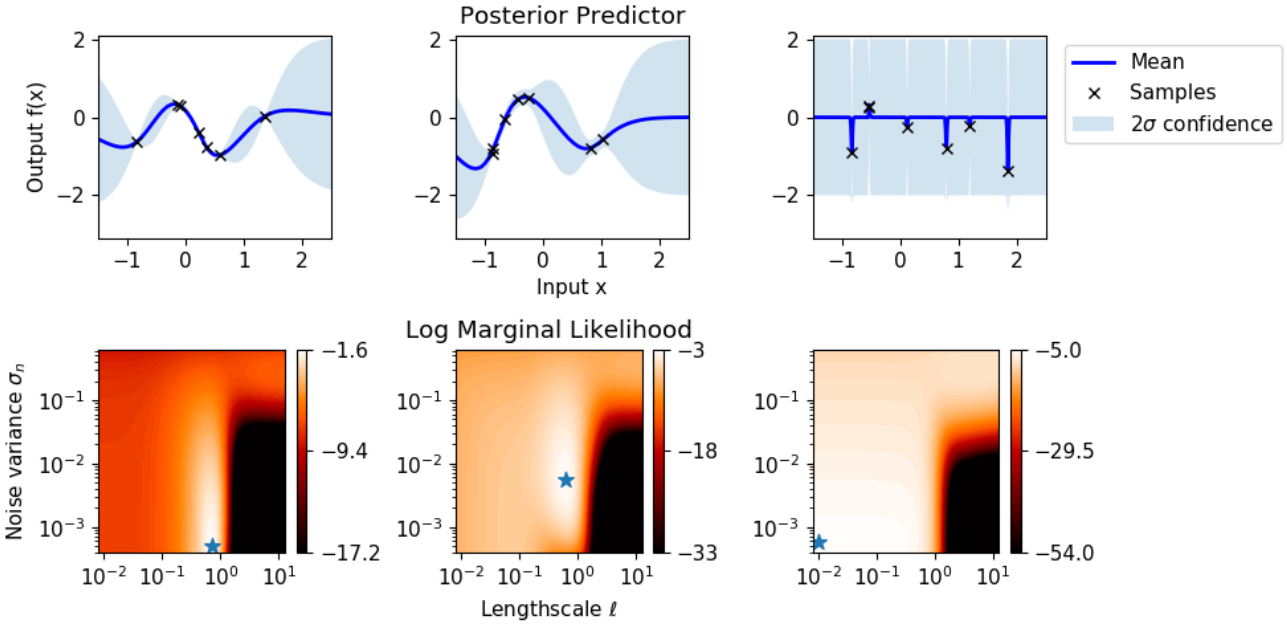
Gaussian Process - Upper Confidence Bound
The Upper Confidence Bound (UCB) for a Gaussian process [30] is defined as

$$\alpha_{\text{UCB}}(\mathbf{x}|\mathscr{D}) = \mu(\mathbf{x}) + \beta\sigma(\mathbf{x}),$$

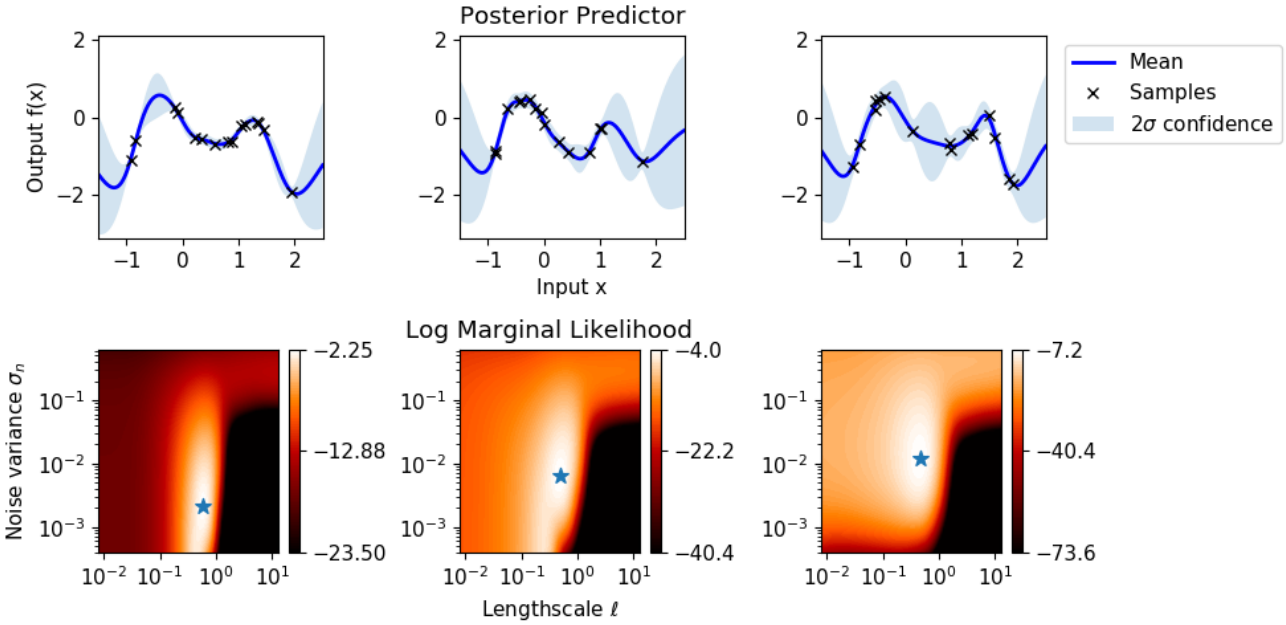
where β is a hyperparameter that regulates the trade-off between exploitation and exploration and has to be specified by the user.
In Figure 3.2, the presented acquisition functions are visually compared on the previously introduced 1-dimensional toy problem (3.9). In our experiments, we will use the EI acquisition as it requires no further hyperparameter and is generally less exploiting than PI [26]. For a broader analyses of acquisition function in the context of Bayesian optimization, including information-based criterion's and using portfolios of acquisition functions, see [27].



(a) Gaussian process hyperparameter optimization using the marginal likelihood for three random samples.

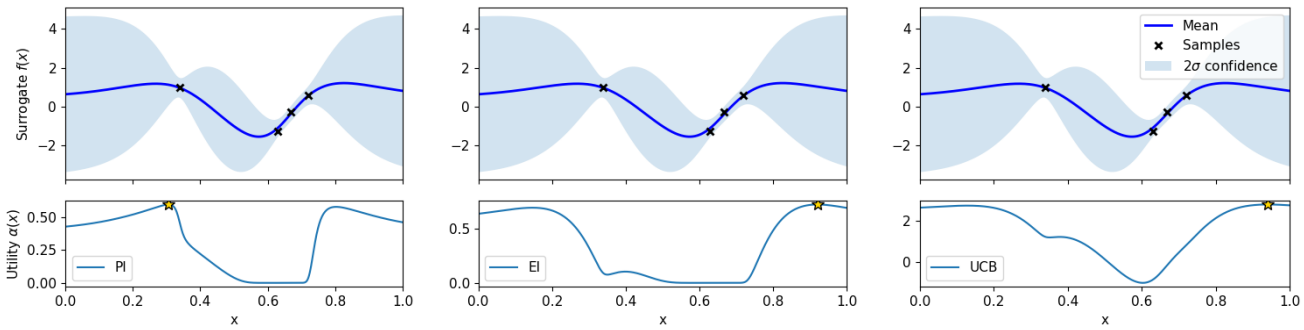


(b) Gaussian process hyperparameter optimization using the marginal likelihood for seven random samples.

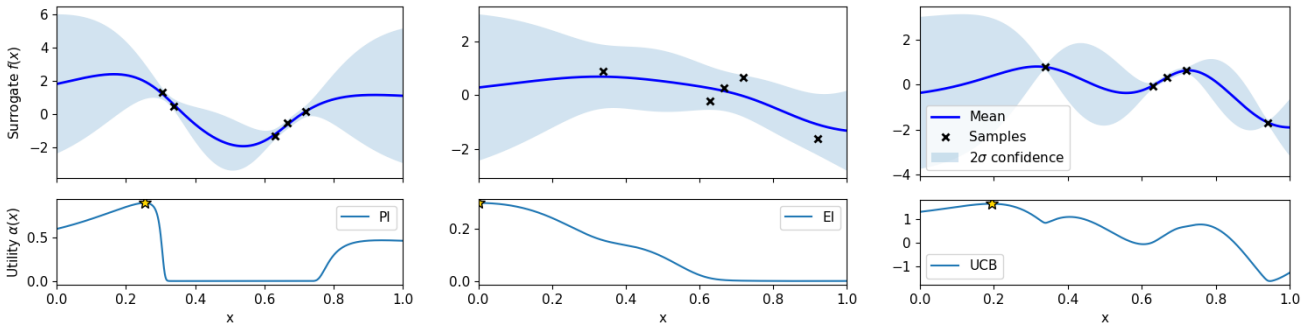


(c) Gaussian process hyperparameter optimization using the marginal likelihood for fifteen random samples.

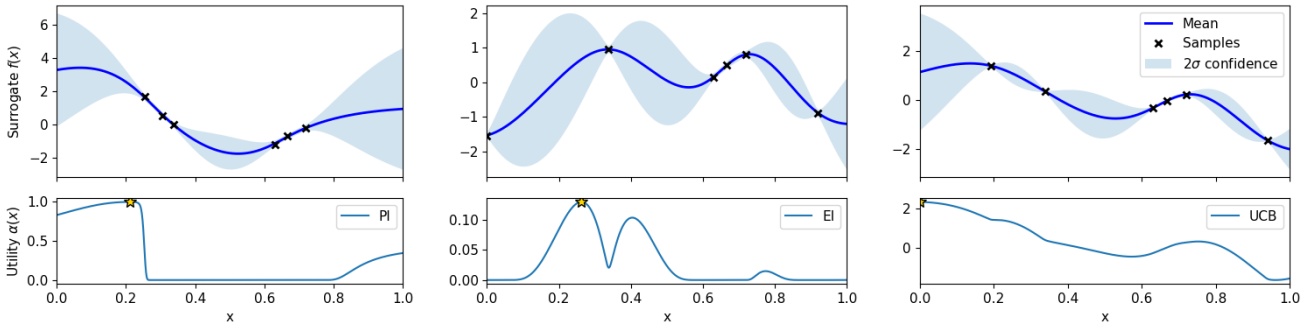
Figure 3.1: Visualization of the Gaussian process hyperparameter optimization using the marginal likelihood. The Gaussian process uses the Matérn kernel with a fixed signal variance $\sigma_f = 1.0$. The three columns present different random seeds, which result in different sample location. The subfigures show how the marginal likelihood surface evolves for increasing number of samples. The upper row of each subfigure visualizes the posterior prediction of the Gaussian process for the optimized hyperparameters. The lower row of each subfigure visualizes the logarithm of the marginal likelihood as function of the noise variance σ_n and length-scale ℓ. Likelihoods with a value below ten times the value of the maxima are clipped to have better visualization results. The star indicates the found maximum and corresponds to the visualized posterior model.



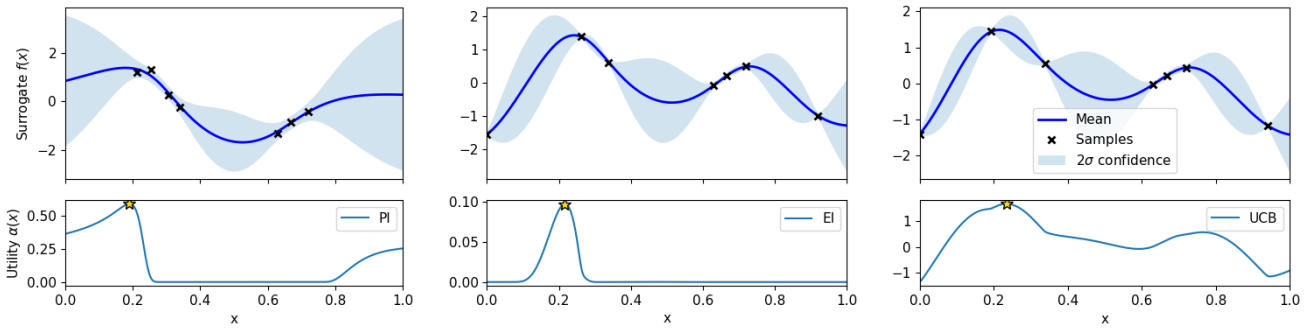
(a) Iteration 1



(b) Iteration 2



(c) Iteration 3



(d) Iteration 4

Figure 3.2: Comparison of the introduced acquisition functions PI, EI and UCB on the 1-dimensional toy function (3.9). The columns present the three acquisition criterions while the rows show the progression of Bayesian optimization for four iterations, starting with four random samples. For the surrogate model, we use a Gaussian process with Matérn kernel, inputs that are normalized to $(0, 1)$ and standardized observations y.

4 Method

One of the main challenges for the success of domain randomization is the design of the simulation parameter distribution $p_\psi(\xi)$. Typically, the distribution $p_\psi(\xi)$ covers the uncertainty about the underlying parameters of the real world environment. This uncertainty is often based on system identification as well as human expertise. Ideally, the real world dynamics are covered by the chosen distribution. In practice, designing this distribution based on prior belief can be error-prone and most likely there will be physical effects which are not covered by the model [10, 18]. If a policy, found by handcrafted domain randomization, fails on the real system, often trial-and-error is used to find a distribution where learned policies can be successfully transferred. This rises the problem of automating the adapting of the distribution over the domain parameters.

In the following, we present a data-driven approach to solve the automation problem. The problem of adapting the distribution's hyperparameters using data from the real world system has already been subject of the state-of-the-art research [10, 18, 17]. These works are build upon the same idea as probabilistic model learning, often seen in model-based reinforcement learning. The observations received from executing learned policies on the real system are used to adapt the hyperparameters of the distribution such that the observations generated in simulation match the observations from the real world.

However, we want to investigate a more direct approach, similar to Vuong et al. [19]. The performance of a policy is quantified by the expected return on the real-world system. Instead of optimizing the similarity between the state transitions of source and target domain, we aim for directly optimizing the return on the real-world system. Our goal is to find the distribution p_ψ which maximizes the expected return on the target system $J^{\text{real}}(\pi_\theta(\psi))$. Thus, we desire to find

$$\psi^* = \underset{\psi}{\operatorname{argmax}}\, J^{\text{real}}\big(\pi_\theta(\psi)\big), \tag{4.1}$$

where the policy parameters θ are found by solving the inner optimization problem (3.3). Under the assumption that policies trained for a specific value of ψ, but different random seeds, lead to a similar performance on the target system, the problem (4.1) can simplify by removing the dependency on θ

$$\psi^* = \underset{\psi}{\operatorname{argmax}}\, J^{\text{real}}(\psi). \tag{4.2}$$

This simplification can further be motivated by our high-level goal of finding a source distribution over the domain parameters, specified by ψ, that enables reliable training of policies that are transferable to the target system. Therefore, the approach presented in this thesis differs from the typical setting where one tries to find one configuration of policy parameters which solves the task.

In the following we want to explore how Bayesian optimization can be used to efficiently solve (4.2). Considering the very expensive evaluation of (4.2) and a sufficiently low dimensional domain parameter set ψ, Bayesian optimization is well suited for this problem. Furthermore, Bayesian optimization can handle noisy function evaluations which in this case corresponds to the difference in performance between different policies trained for the same hyperparameters ψ.

Algorithm 3 presents a high-level view on how Bayesian optimization can be used to solve the outer optimization problem (4.2) in combination with any reinforcement learning algorithm like PPO solving the inner optimization problem (3.3).

Algorithm 3 Bayesian optimization for learning the domain parameter distribution

1: **Input**: boundaries for ψ, initial dataset $\mathcal{D}_0$
2: **for** $n = 0, 1, 2, \ldots,$ **do**
3: $\quad \pi_\theta(\psi_n) \leftarrow$ *PolicyOptimizer*
4: $\quad J^{\text{real}}(\psi_n) \leftarrow$ Run policy $\pi_\theta(\psi_n)$ on the target system
5: $\quad \mathcal{D}_{n+1} = \{\mathcal{D}_n, (\psi_n, J^{\text{real}}(\psi_n))\}$
6: $\quad \psi_{n+1} \leftarrow$ Bayesian optimization step using $\mathcal{D}_{n+1}$
7: **end for**

5 Experiments

In this chapter, we present empirical results to the proposed method of optimizing the domain parameter distribution through Bayesian optimization. We compare the sim-to-real performance of policies learned from the distributions found by Bayesian optimization to the policies learned from handcrafted distributions and exact models. In the remainder, the exact model and the handcrafted randomization are referred to as the baselines. The goal is to empirically evaluate, whether the domain parameter distribution can be optimized such that policies trained on these distributions are more likely to be transferable to the real system. We quantify the transferability with the expected average reward from the real-world system.

5.1 Platform

To benchmark the experiments, we chose the Quanser Qube, see Figure 5.1a, which is a Furuta pendulum (also called rotary inverted pendulum). The Quanser Qube is a robust to use platform and therefore well suited for benchmark testing. The Furuta pendulum was first introduced by Furuta et al. [31] in 1992. It consists of a freely rotating pendulum attached to a motor driven pole. The motor driven pole is rotatable in the horizontal plane and the pendulum can swing in a vertical plane orthogonal to the arm. By having two degrees of freedom but only one actuator it is an underactuated system. The dynamics can be derived using the Euler-Lagrange method, presented in [31]. The goal is to swing the non-actuated pendulum up and stabilize around the upright unstable equilibrium. This is a typical benchmark test used in control theory and robotics.
The state space of the Furuta pendulum is given as $\mathbf{s} = [\theta, \alpha, \dot{\theta}, \dot{\alpha}]$, where θ is the angle of the rotary pole and α the angle of the pendulum, see Figure 5.1b. Its action space is given as $\mathbf{a} = [V_m]$, where V_m is the motor voltage, which induces a momentum to the rotary pole. For the Quanser Qube, the state and action space are constraint to

$$|\mathbf{s}| < [2.3,\ 4\pi,\ 30,\ 40], \qquad |\mathbf{a}| < [5].$$

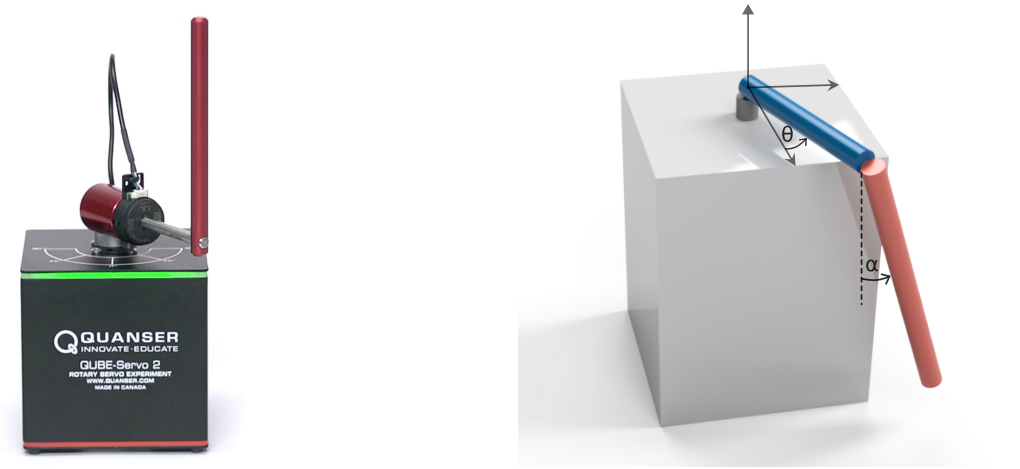
The constraints on α and the velocities are not hardware constraint but are used to ensure safe behavior of policies on the real system.
The system dynamics are given as

$$\begin{aligned}
&\left(m_p L_r^2 + \frac{1}{4} m_p L_p^2 - \frac{1}{4} m_p L_p^2 cos(\alpha)^2 + J_r\right)\ddot{\theta} + \left(\frac{1}{2} m_p L_p L_r cos(\alpha)\right)\ddot{\alpha} + \\
&\left(\frac{1}{2} m_p L_p^2 \sin(\alpha)\cos(\alpha)\right)\dot{\theta}\dot{\alpha} - \left(\frac{1}{2} m_p L_p L_r \sin(\alpha)\right)\dot{\alpha}^2 + D_r\dot{\theta} = \frac{k_m(V_m - k_m\dot{\theta})}{R_m}, \\
&\left(\frac{1}{2} m_p L_p L_r \cos(\alpha)\right)\ddot{\theta} + \left(J_p + \frac{1}{4} m_p L_p^2\right)\ddot{\alpha} - \left(\frac{1}{4} m_p L_p^2 \cos(\alpha)\sin(\alpha)\right)\dot{\theta}^2 + \frac{1}{2} m_p L_p g \sin(\alpha) + D_p\dot{\alpha} = 0
\end{aligned}$$

and are used to deploy the randomizable simulator. The physics parameters and constants are detailed in Table 5.1.
The goal position for the presented benchmark test is given as $\mathbf{s}_{\text{des}} = [0, \pi, 0, 0]$ and $\mathbf{a}_{\text{des}} = [0]$. The reward is defined as a squared exponential function on the error of state and action towards the goal

$$r(\mathbf{s}, \mathbf{a}) = \exp\Big((\mathbf{s} - \mathbf{s}_{\text{des}})\ \mathbf{Q}\ (\mathbf{s} - \mathbf{s}_{\text{des}}) + (\mathbf{a} - \mathbf{a}_{\text{des}})\ R\ (\mathbf{a} - \mathbf{a}_{\text{des}})\Big),$$

with $\mathbf{Q} = \text{diag}(0.2, 1, 0.02, 0.005)$ and $R = 0.003$.
Further, the control frequency is set to 100Hz and the maximum trajectory length to 6 seconds, resulting in 600 state transitions for a full trajectory. However, a trajectory is terminated early if the boundary conditions are violated.



(a) Qube by Quanser [32] **(b)** Mechanical model of the Furuta pendulum

Figure 5.1: Furuta pendulum benchmark platform

Description	Abbreviation	Nominal value	Unit
Gravity	g	9.81	m/s^2
Motor resistance	R_m	8.4	Ω
Motor back-end constant	k_m	0.042	$V \cdot s$
Rotary pole damping	D_r	5×10^{-6}	$N \cdot m \cdot s$
Pendulum damping	D_p	10^{-6}	$N \cdot m \cdot s$
Rotary pole mass	M_r	0.095	kg
Pendulum mass	M_p	0.024	kg
Rotary pole length	L_r	0.085	m
Pendulum length	L_p	0.129	m
Rotary pole inertia	J_r	$M_r \times L_r^2/12$	$kg \cdot m^2$
Pendulum inertia	J_p	$M_p \times L_p^2/12$	$kg \cdot m^2$

Table 5.1: Quanser Qube domain parameters and constants

5.2 Setup

In this section, we present the experimental setup for testing the baseline experiments and our method to compare them in terms of their sim-to-real performance. As baselines experiments are considered learning a policy from a nominal model and learning a policy from an exact handcrafted domain randomization distribution. For the exact (or nominal) model, the nominal parameters reported in Table 5.1 are used without any randomization. Next, two randomized models are deployed by extending the nominal with a handcrafted randomization over (i) the masses and lengths (M_r, M_p, L_r, L_p) while using the nominal parameters for the remaining domain parameters and (ii) all domain parameters (the inertia's are treated as constants, see Table 5.1). We choose a normal distribution where the mean is equal to the nominal value of the corresponding domain parameter and the standard deviation is equal to a fifth of the identified value $p_\psi(\xi) = \mathcal{N}(\xi_{\text{nom}}, \xi_{\text{nom}}/5)$.
Our method proposes that the domain parameter distribution can be optimized with respect to the performance on the real-world system, see Algorithm 3. In the following, we outline the setup of the optimization process and than compare the found optimized distribution to the baselines in terms of sim-to-real performance.
A typical choice for the distribution type is the normal distribution like we used for the handcrafted distributions. Using normal distributions results in two hyperparameters per domain parameter, the mean and the standard deviation. Even the Furuta pendulum as a benchmark systems has nine domain parameters (neglecting the inertia's), which would render Bayesian optimization with optimizing in an 18-dimensional search space. Although successful applications for up to 20-dimensional inputs have been demonstrated, we desire to benchmark our method on a lower dimensional task. Therefore, we chose to optimize the distribution over a subset of the domain parameters. By that, the necessity of optimizing the distribution over every domain parameter can also be analyzed. Concretely, we optimize the distribution of mass and length of both rotary pole and pendulum (M_r, M_p, L_r, L_p), rendering Bayesian optimization with an 8-dimensional search space. The boundaries used for our experiments are presented in Table 5.2. While optimizing masses and lengths, we fixed the remaining domain parameter (g, R_m, k_m, D_r, D_p) to (i) the handcrafted distribution and (ii) the nominal domain parameter values. As acquisition function we use EI (3.10) and for the Gaussian process we use a zero prior mean and the Matérn kernel with smoothing factor $5/2$ (3.7). As initialization we use 15 random samples. Further, inputs to the Gaussian proccess are normalized to the unit hypercube and output are standardized to make use of priors on the kernel hyperparameters for marginal likelihood optimization. We implemented our Algorithm 3 using Botorch, a library for Bayesian optimization build upon GPyTorch [33], which itself is an implementation of Gaussian processes in PyTorch [34].
For the inner optimization problem of learning a policy given a simulator, we use PPO. To ensure comparability, we use the same hyperparameter setting in each experiment. The policy π and advantage function estimate $\hat{A}$ are represented as feed-forward neural networks with two hidden layers of size 64 and tanh non-linearity. To update the parameters of the neural networks, we use a learning rate of 5×10^{-4}, 10 optimization epochs for each estimator update and set the corresponding estimator sample size to 64. Further hyperparameter are the temporal discount factor of $\gamma = 0.99$, the trade-off factor for bias and variance of $\lambda = 0.95$ and the clipping factor of $\varepsilon = 0.1$, see [3]. Finally, we use 100 policy updates, each using 48000 state transitions, which corresponds to at least 80 trajectories. Such high numbers of samples per update are not required to learn good policies with nominal models. However, for domain randomization it ensures that the randomly sampled parameters represent the underlying distribution to lower bias.
We have specified five different models that we now can compare against each other in terms of their sim-to-real performance:

- nominal parameter values,
- handcrafted randomization for all parameters,
- handcrafted randomization for masses and lengths and nominal value for the remaining parameters,
- optimized randomization for masses and lengths and handcrafted randomization for the remaining parameters,
- optimized randomization for masses and lengths and nominal value for the remaining parameters.

To ensure that the sim-to-real performance comparison of the presented methods is not disturbed by the random seed, we train twenty policies over different random seeds on each model. Then, all trained policies are tested on the real system and their average reward is used as measure for the sim-to-real performance of the model.

	μ_{M_p}	σ_{M_p}	μ_{M_r}	σ_{M_r}	μ_{L_p}	σ_{L_p}	μ_{L_r}	σ_{L_r}
Lower	0.019	5e-5	0.085	3e-4	0.115	5e-4	0.070	2e-4
Upper	0.029	5e-3	0.110	3e-2	0.140	5e-2	0.095	2e-2

Table 5.2: Boundaries for the search space of Bayesian optimization. The columns present the means μ and standard deviations σ of the normal distribution over each domain parameter. The rows specify the corresponding lower and upper boundaries of the search space.

5.3 Results

In this section, we present our empirical results and evaluate these. The proposed method of optimizing the domain parameter distribution through Bayesian optimization converged for both experiments after 10 iterations, given 15 initial random samples. The optimized distributions are then given as the inputs that maximize the posterior mean of the Gaussian processes and are presented in Figure 5.2. The plots show a comparison of the optimized distributions, the handcrafted distribution and the nominal values for the masses and lengths of the Furuta pendulum. Subsequently, we evaluated the performance of the five models on 20 random seeds, resulting in 20 different policies, for a thorough estimation of performance that is not biased by random seeds. The training progress is presented in Figure 5.3. It is noticeable that all training runs converge to similar good solutions in terms of their average reward in simulation. For each training run, the policy yielding the best average reward is subsequently transfered to the real-world system. The sim-to-real performance results are reported in Table 5.3. As it can be seen, trained policies with the nominal model have the worst average sim-to-real performance despite having the best results in simulation. For the models with handcrafted randomization and combination of handcrafted randomization and nominal values, we see a clear increase in their average sim-to-real performance. However, the clearly best result in terms of sim-to-real performance was achieved by the models with the optimized randomization found by our proposed method. The policies trained on the optimized domain parameter distribution have significantly higher probability of being transferable to the real-world system than the policies trained on the handcrafted randomization. The experiment demonstrates that we are able to optimize the domain parameter distribution with respect to the policies real-world performance. Further, the results suggest, that it is not generally necessary that we randomize every possible domain parameter. For the Furuta pendulum swing-up task, randomizing the masses and lengths with a handcrafted distribution showed similar to slightly better results in terms of transferability then randomizing all domain parameters. The same phenomena can also be observed by comparing policies over partial optimized randomization with handcrafted randomization and partial optimized randomization with nominal parameters. Both strategies result in a similar sim-to-real performance. Subsequently, we do not generally need to optimize the distribution over all domain parameters to get a model that allows reliable training of transferable policies. It should be emphasized that a difference in success rate of 5 %, see 5.3, here corresponds to a single policy more being successfully transferable. However, the differences in success rate between nominal model, handcrafted randomization and optimized randomization are distinctively large and therefore highly unlikely caused by statistical imprecision.
For the nominal model, despite all policies having a similar high reward in simulation, only a small percentage (10 %) of the policies were transferable while others were not. This result suggests that (i) policies can easily exploit errors in the nominal model and become "over-optimized" which leads to policies that typically do not generalize to the real-world system (ii) the transferability of a policy can greatly vary depending on the found local optimum in the policy parameters. The policy parameters itself only depend on the chosen random seed, as the model and hyperparameters are fixed. It is worth noting, that the failed policies all terminated early by violating the state boundaries.
Typically, we want to randomize domain parameter we are uncertain about. Parameters like gravity, masses and lengths can be measured very accurately. As seen in this experiment, it can still make sense to randomize these parameters to compensate for errors in the model and making model errors less exploitable.
Further, we also tested uniform distributions, but they showed worse performance in terms of sim-to-real performance, and a deeper analysis about the influence of the distribution type could be subject to future research.

	Simulation		Real-world system		
	Mean reward	Standard deviation	Mean reward	Standard deviation	Success rate
Nominal model	541	8.5	141	192	10 %
Handcrafted randomization	453	26.3	311	235	55 %
Combination of nominal model and handcrafted randomization	487	26.1	324	232	60 %
Combination of optimized randomization and handcrafted randomization (Opt/Hc)	499	21.2	454	123	85 %
Combination of optimized randomization and nominal model (Opt/Nom)	514	16.4	476	110	90 %

Table 5.3: Sim-to-real performance comparison between baselines and the optimized distribution found by our method. The baselines are the nominal (exact) model, a handcrafted randomization over all domain parameters and a combination, where only masses and lengths are randomized and the remaining parameters are fixed to their nominal value. With our method, we optimized the distributions over the masses and lengths while once using the handcrafted randomization for the remaining parameters (Opt/Hc) and once using the fixed nominal values for the remaining parameters (Opt/Nom). The optimized distributions are shown in Figure 5.2. For the experiment, 20 policies were trained until convergence for each evaluated method, see Figure 5.3. Additionally, we report the success rate which is a subjective quantity. For this benchmark test, we call a policy successful if it is able to stabilize the pendulum at the upright position, does not terminate early (e.g. by violating the state boundaries) and has a reward of at least 400.

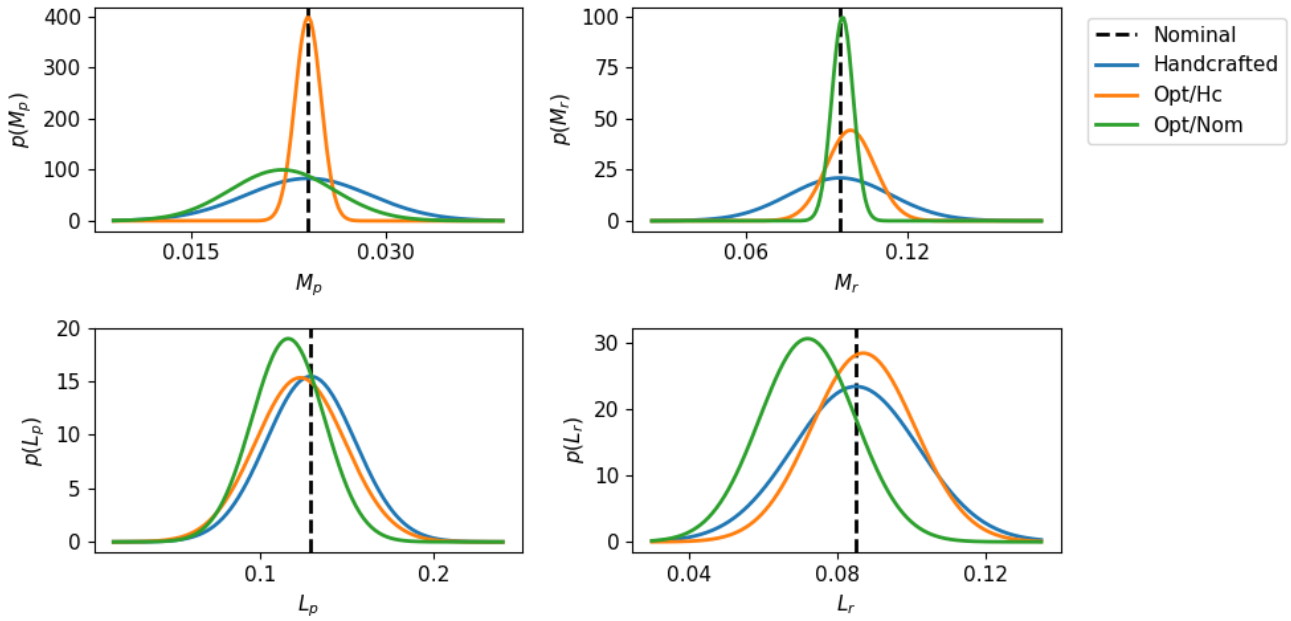
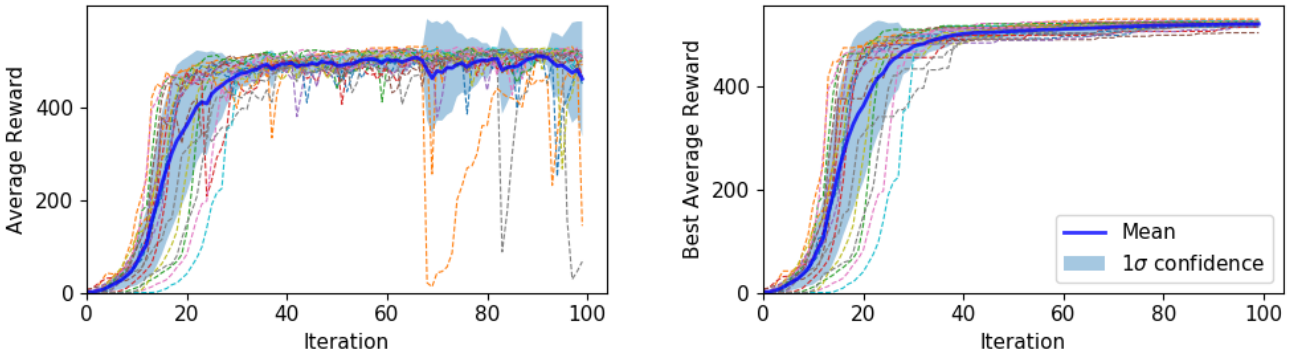
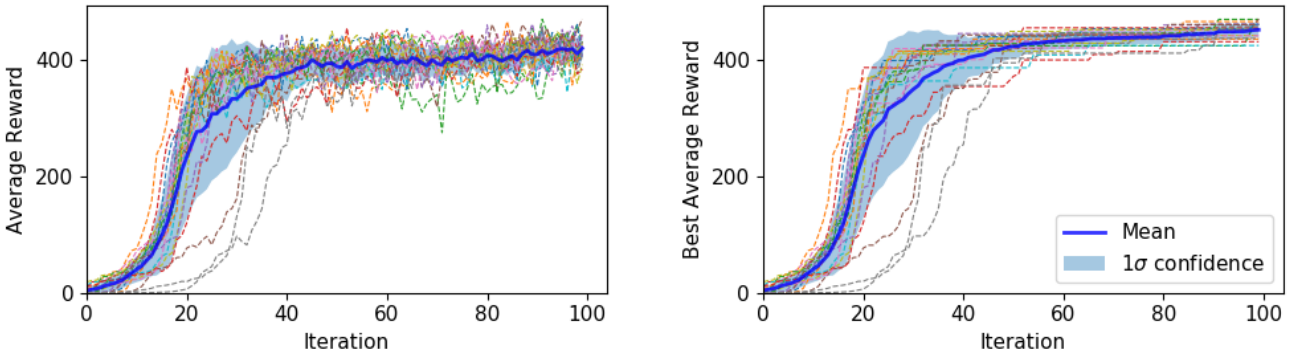


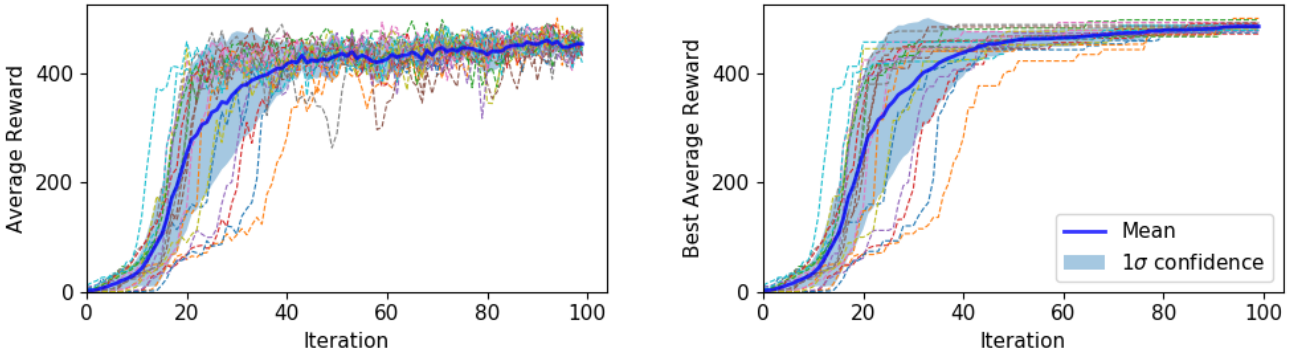
Figure 5.2: Comparison of the domain parameter distributions over the masses and lengths of the Furuta pendulum for the evaluated methods. The nominal values corresponds to the values in Table 5.1. For the handcrafted distribution, we use a normal distribution with the nominal value as mean and a standard deviation equal to a fifth of the nominal parameter value. The abbreviation 'Opt/Hc' stands for the optimized distribution over the masses and lengths while using the handcrafted randomization for the remaining parameters. 'Opt/Nom' stands for the optimized distribution over the masses and lengths while using nominal values for the remaining parameters.



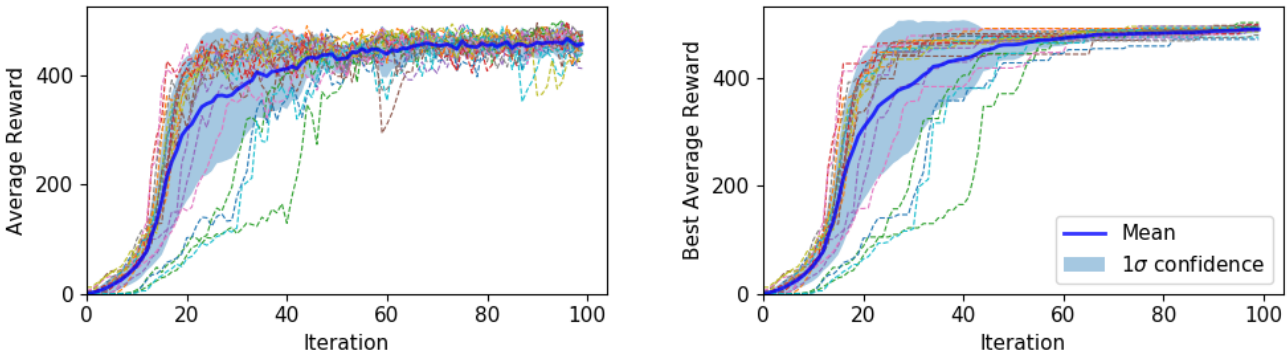
(a) Nominal model.



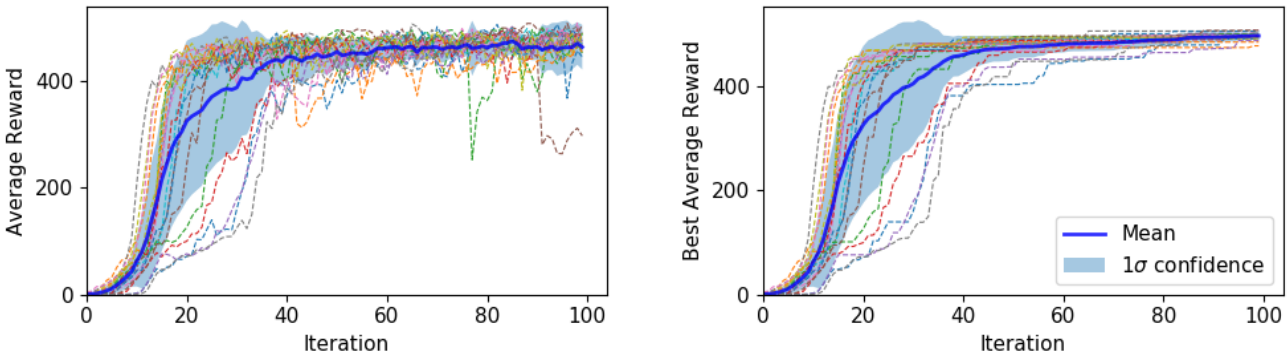
(b) Model with handcrafted randomization.



(c) Model with handcrafted randomization for the masses and lengths and nominal values for the remaining domain parameters.



(d) Model with optimized randomization for the masses and lengths and handcrafted randomization for the remaining domain parameters.



(e) Model with optimized randomization for the masses and lengths and nominal values for the remaining domain parameters.

Figure 5.3: Training progress of PPO for learning policies from different models. The plots in the left column shows the average reward over the current policy. The plots in the right column presents the average reward of the currently best policy. Each plot shows the progress of 20 policies trained on different random seeds as well as their average reward over the iterations.

6 Conclusion

In this thesis, we showed how domain randomization can be used to learn policies in simulation that are more likely to be transferable to real-world system compared to policies which learned from exact models. One of the decisive factors in domain randomization is the domain parameter distribution design. When policies trained on a simulator with handcrafted randomization based on prior belief fail on the real-world system, trial-and-error is commonly used to reshape the domain parameter distribution. We argue that this task can be automated. Therefore, we proposed to use Bayesian optimization to iteratively optimize the domain parameter distribution with respect to the policy performance on the real-world system.

Our experiments demonstrate that Bayesian optimization can be used to find effective domain parameter distribution in just a few iterations. An effective distribution here means, that policies trained under this randomization are more likely transferable to the real-world system than policies trained on a naive, handcrafted randomization. We observered a substentially increased success rate for policies trained from the optimized distributions found by our proposed method then policies trained from handcrafted randomization.

By modeling the relation between distribution and real-world performance by a Gaussian process, we assumed that the real-world performance for policies trained on different seeds result in Gaussian distributed real-world rewards. However, in our experiment, policies trained on the same distribution but different random seeds typically result in failure (very low reward) or success (very high reward). Therefore, Bayesian optimization has no guarantees of converging for this optimization problem, but nevertheless showed promising results.

In the future, we are interested in how other global optimization algorithms compare to our approach for optimizing domain parameter distributions. Another interesting field for future research could be, how Bayesian optimization can be used in a context where prior access to the real system is not possible. Furthermore, methods for optimizing domain parameter distributions only in simulation combined with approaches like ours, that use real-world data, could further decrease real-world data requirements for successful sim-to-real transfer.

Bibliography

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015.

[2] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, pp. 1329–1338, 2016.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[4] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3389–3396, IEEE, 2017.

[5] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *European Conference on Artificial Life*, pp. 704–720, Springer, 1995.

[6] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," 2018.

[7] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," 2018.

[8] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8, IEEE, 2018.

[9] J. Tobin, L. Biewald, R. Duan, M. Andrychowicz, A. Handa, V. Kumar, B. McGrew, A. Ray, J. Schneider, P. Welinder, *et al.*, "Domain randomization and generative models for robotic grasping," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3482–3489, IEEE, 2018.

[10] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," 2018.

[11] M. Mozifian, J. C. G. Higuera, D. Meger, and G. Dudek, "Learning domain randomization distributions for transfer of locomotion policies," 2019.

[12] L. Ljung, "System identification," *Wiley Encyclopedia of Electrical and Electronics Engineering*, 2001.

[13] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.

[14] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, IEEE, 2017.

[15] F. Muratore, F. Treede, M. Gienger, and J. Peters, "Domain randomization for simulation-based policy optimization with transferability assessment," in *Conference on Robot Learning*, pp. 700–713, 2018.

[16] K. Lowrey, S. Kolev, J. Dao, A. Rajeswaran, and E. Todorov, "Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system," in *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pp. 35–42, IEEE, 2018.

[17] F. Ramos, R. Carvalhaes Possas, and D. Fox, "Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators," 2019.

[18] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, "Epopt: Learning robust neural network policies using model ensembles," 2016.

[19] Q. Vuong, S. Vikram, H. Su, S. Gao, and H. I. Christensen, "How to pick the domain randomization parameters for sim-to-real transfer of reinforcement learning policies?," 2019.

[20] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, "Active domain randomization," 2019.

[21] S. Paul, M. A. Osborne, and S. Whiteson, "Fingerprint policy optimisation for robust reinforcement learning," in *International Conference on Machine Learning*, pp. 5082–5091, 2019.

[22] W. Yu, C. K. Liu, and G. Turk, "Policy transfer with strategy optimization," 2018.

[23] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[24] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[25] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," 2010.

[26] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, pp. 2951–2959, 2012.

[27] 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, 2015.

[28] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*, pp. 63–71, Springer, 2003.

[29] 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.

[30] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," 2009.

[31] 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.

[32] Quanser Inc., "Qube - servo 2 - quanser."

[33] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson, "Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration," in *Advances in Neural Information Processing Systems*, pp. 7576–7586, 2018.

[34] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.