Combining Simulations and Real Experiments for Sample-efficient Gait Parameter Learning using Multi-Fidelity Entropy Search

Bachelor Thesis

Author:
André Brandenburger

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
Prof. Dr. Maren Bennewitz

Autonomous Intelligent Systems (AIS)
Computer Science Institute VI
University of Bonn

Bonn, Germany
April 14, 2018

Eidesstattliche Erklärung nach BAPO 2011 §17 Abs. 7

Hiermit erkläre ich, dass ich diese Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe. Die Arbeit habe ich bisher weder veröffentlicht, noch einem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt.

(Ort, Datum)	(Unterschrift)

Zusammenfassung

Der stabile Gang auf zwei Beinen ist eine anspruchsvolle Aufgabe und erfordert meist wiederholte Anpassung der Parameter aufgrund von Abnutzung, Anwendungsgebiet oder Ähnlichem. Auch wenn der Einsatz von Fused-Feedback den Gang bereits stabilisiert, hängt dessen Leistungsvermögen stark von den neu eingeführten Parametern ab. Diese Parameter korrekt einzustellen erfordert oft viel Zeit und Expertenwissen, während es zudem die Hardware des Roboters abnutzt. Dieses Problem kann zwar teils gelöst werden, indem der Roboter simuliert wird, jedoch spiegeln diese Simulationen nur schlecht die echte Welt wider. Unter Anwendung von stichprobeneffizienter Bayes'scher Optimierung möchten wir in dieser Arbeit die Gang-Parameter lernen um die Arbeitslast und das nötige Expertenwissen zu reduzieren. Außerdem wird der Ansatz der Multi-fidelity Entropy Search auf dieses Problem angewandt um Simulationen effizient in den Optimierungsprozess einzubinden. Schließlich wird diese Methode auf die igus® Humanoid Open Platform angewandt und die resultierenden Ergebnisse werden analysiert.

Abstract

Bipedal walking is a challenging task, whose performance strongly depends on the choice of parameters. While the Fused Feedback Mechanisms improve the gait stability, they introduce new parameters to be tuned. Generally, this process of tuning not only requires time and expert knowledge, but also wears off the hardware of the robot. While this problem can be partially solved by simulating the robot, simulations do not fully represent real-world performance. Using sample-efficient Bayesian Optimization we want to learn gait parameters in order to alleviate the workload and to reduce the required specific knowledge of the operator. Furthermore, using Multi-fidelity Entropy Search, this approach will combine valuable real-world experiments with robot simulations to minimize system wear. In the end, this approach will be applied to the igus® Humanoid Open Platform to evaluate its performance.

Contents

List of Figures

Acknowledgements

First of all I would like to thank Diego Rodriguez and Prof. Behnke for their helpful and creative input and supervision. Additionally, thanks to everybody of the soccer group for giving me assistance whenever I needed it. Finally, I would like to thank Alonso Marco Valle and the Max-Planck-Institute for Intelligent Systems for providing me with their Multi-fidelity Entropy Search implementation.

Chapter 1

Introduction

A fundamental task for humanoid robots is walking robustly. The problem of generating such a gait has already been investigated by numerous approaches, which yielded improvements in aspects like stability[3, 19], speed[8], energy efficiency[16, 12] or accuracy[25, 12].

Nonetheless, many approaches introduce parameters that need to be tuned for specific robot platforms or applications (e.g. ground surface). The process of tuning is by no means trivial and it is time consuming for the robot operator, while it also wears off servos and bearings of the robot. The hardware wear-off does not only cost money, but may also result in invalid parametrizations, since hardware replacements might perform differently. To cope with that problem, experiments can be performed in a simulated environment, which gives a rough estimate about the real-world performance of parameters at a low cost.

However, solely working on simulations will not yield exactly the same results as performing real-world experiments. This point is especially relevant for low-cost robots, since this hardware does not normally work as precise as expected. On the other hand, humanoid robotics is steadily making its way into our lives, where low-cost robots are significantly more appealing. Additionally, cheaper costs give institutions with small funding the opportunity to perform research on the topic.

This thesis will address the problem of gait parameter learning with focus on sample efficiency, while trading off real-world experiments with simulations to alleviate the workload of the robot operator and preserve the hardware. We will apply Multi-fidelity Entropy Search (MF-ES)[18] on parameters of the NimbRo gait[3]. Using relative entropy as a quantitative measure, MF-ES chooses the most informative point at each iteration, which reduces the amount of required evaluations(see Fig. 1.1). Since MF-ES resembles a Reinforcement Learning approach, it is necessary to define a customized cost function to evaluate the parameter performance. This cost function can be arbitrarily complex by incorporating different criteria that are needed to yield a satisfactory gait. In this manner, we propose to use a cost function based on the Fused Feedback as a measure of gait stability. Finally, this approach will be evaluated on the igus® Humanoid Open Platform[4].

Consequently, we will talk about the related work in Chapter 2. Next, we shortly explain the NimbRo gait in Chapter 3. Subsequently, the method of Multi-fidelity Entropy Search will be defined in Chapter 4. In Chapter 5, this

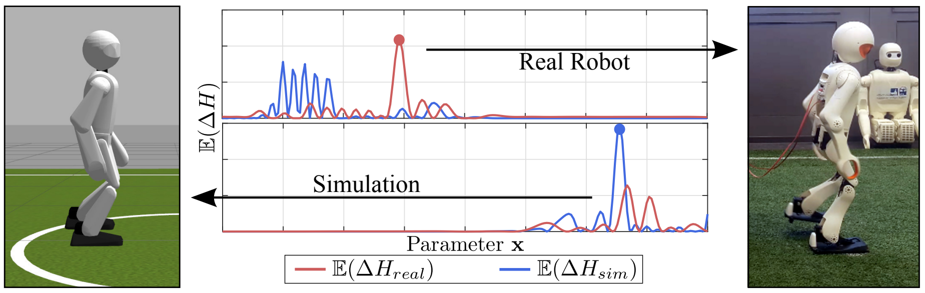

Figure 1.1: The proposed algorithm will trade off simulations with real experiments by maximizing relative Entropy.

method is applied on gait parameter learning by constructing a cost function based on the robots gait stability. Moreover, the problem of simulator noise is addressed and a regularization is introduced. Additionally, we propose a termination criterion based on the Kullback-Leibler-Divergence. In the end, the experimental results will be presented in Chapter 6.

Chapter 2

Related Work

In this Chapter we will firstly address bipedal walking, then we will discuss different approaches of gait optimization.

There are numerous methods to produce stable gaits, but by far the most common approaches are based on the calculation of the Zero-Moment Point (ZMP). As a criterion of stability, the ZMP, the point where the horizontal component of the moment of the ground reaction force is zero, is utilized to calculate the optimal gait trajectory[15]. The application of ZMP methods is especially successful on robots with actuators of high quality since it requires exact joint and torque measurements. Additionally, it can only be used on robots that are equipped with foot force sensors. Due to its reputation, it is used in prominent examples like the Honda P2[14], Atlas[26] and TORO[9].

In contrast to ZMP approaches, open-loop methods typically do not require exact joint measurements or additional external sensors. For example, the NimbRo gait uses a motion pattern generator to produce periodical gait motions, which depend on a given velocity vector[22]. This vector includes omnidirectional linear motion and also supports a target rotational velocity. Together with compliant actuator settings, this method is able to produce a stable gait, but its performance is not satisfactory regarding recovery from external disturbances[22].

Thus, this open-loop approach was extended with Fused Feedback Mechanisms[3] (see Chapter 3). This extension of the open-loop gait was able to yield improvements, but its performance highly depends on a gain matrix, which can be different for each robot. Hence, the matrix needs to be optimized by the robot operator so the method can perform well.

The problem of parameter tuning has already been observed previously and was addressed by applying learning algorithms to find optimized values. Röfer suggests using evolutionary algorithms as a solution for a Sony Aibo quadruped robot[25]. This optimization involves 26 parameters and uses a fitness function based on smoothness and accuracy of the gait. Since the author does not provide information about the required number of iterations, the sample-efficiency of this method remains questionable.

Missura et al. propose an approach to learn foot placement for a stable and distinguished gait[21, 20]. Since these goals can be contradictory, a trade-off is introduced. To ensure that the algorithm does not slow down during the optimization process, the authors chose the Locally Weighted Projection

Regression as a local optimization framework. This decision is particularly important for the online application of the method, since the amount of provided data is high and processing time has to be kept minimal, but leads to the restriction to local minima.

Faber et al. used Policy Gradient Reinforcement Learning and Particle Swarm Optimization to improve an initial hand tuned set of parameters, namely the P-Gain of a foot angle controller and a gait phase resetting mechanism[10]. With modified versions of the algorithms, the robot learned to walk notably faster after 1000 iterations. Although this approach has shown good results, the large amount of iterations needs to be improved in order to preserve the robots hardware.

Heijmink et al. propose a method to learn gait parameters and impedance profiles for a quadruped robot. Moreover, this goal is accomplished by using the PI^2 algorithm with a custom cost function. This cost function consists of the weighted sum of partial cost terms based on speed tracking, energy consumption, joint limits and torques. Using this approach, it was possible to learn problem specific parameters in only 10 minutes of learning. Nonetheless, this approach only features local optimization, since the PI^2 algorithm only explores by corrupting best guesses with Gaussian noise.

A different method proposed by Calandra et al. uses Bayesian Optimization on a bipedal robot[5]. The goal of this approach was to replicate a given target trajectory with the bipedal robot Fox. In that approach the gait was generated using a Finite State Machine (FSM) and 8 parameters of the FSM were optimized. After 80 iterations the algorithm found an optimum that resulted in a stable gait and thus proved to be sample efficient. Still, since it only utilizes real experiments, this approach will lead to hardware wear-off.

Furthermore, [23] also proposes a method based on Bayesian Optimization, which additionally uses information gathered through simulations. They introduce a dimensionality reduction based on the Determinants of Gaits (DoG) to learn a parametrization for higher dimensional controllers. Although this approach was able to achieve good results for a 9-dimensional controller with a remarkably small amount of 20 iterations, it requires, due to its informed kernel structure, a large amount of precomputed simulator data[23].

In contrast to these approaches, [1] suggests to direct the optimization process by using a search distribution to learn parameters on a robotic arm using Bayesian Optimization. This search distribution is a Gaussian restriction on the search space and leads to faster convergence of the algorithm. Still, although this approach dynamically adapts the search distribution, the optimization loses expressibility on a global scope since it only optimizes locally.

In addition to the previous work, Bayesian Optimization has been successfully used in numerous other approaches in the domain of robotics. For instance, Bayesian inference is used in a Gaussian Process framework by Deisenroth et al. to learn control tasks[7]. They applied their approach to a cart-pole system and were able to swing the pole from a state of rest to an upright pose. Additionally, the proposed algorithm tuned the controller of a low-cost robotic arm without pose feedback by incorporating visual feedback.

Furthermore, Berkenkamp et al. propose to use Bayesian Optimization in combination with safety constraints to safely learn parameters for robotic systems. The method uses a combination of safety conditions which are learned during the optimization process of the performance function. With small modifications, the algorithm was able to learn to control a quadrotor UAV.

Moreover, Marco et al. use Bayesian Optimization combined with optimal control to tune an LQR regulator[17]. Using this approach, the authors were able to balance an inverted pendulum by tuning the controller automatically.

Thus, it has already been proven by many past works, that Bayesian Optimization is a suitable tool for sample-efficient learning in robotics. Furthermore, the approach of Bayesian Optimization was improved through the introduction of Entropy Search [13] (see Chapter 4.3), by making it more sample efficient. Additionally, it was extended to use multiple information sources with Multi-fidelity Entropy Search (see Chapter 4.4), which is particularly interesting for robotics, since it is often possible to simulate the respective system.

Similar to some of the discussed approaches, we propose a method based on Bayesian Optimization which utilizes simulations and real experiments to learn parameters for a bipedal gait. In contrary to what was previously presented, our approach trades off real experiments and simulations at each iteration. Moreover, it uses relative Entropy as a measure of information content to lead the process of optimization, which makes this method sample efficient. Furthermore, we propose a termination criterion based on the Kullback-Leibler-Divergence.

Chapter 3

NimbRo gait

In this chapter we will elaborate on the gait that is successfully used in multiple RoboCup competitions by team NimbRo and the extensions that have been made to it.

The gait is based on an open-loop gait pattern generator proposed by Missura et al.}[22]. It is composed of three layers, namely a control interface, a motion pattern generator and a leg interface. The control interface is the first layer of the pipeline, which receives a target velocity vector and preprocesses it by rescaling and smoothing it to prevent discontinuities in the gait, while also providing a gait phase to the next layer. Secondly, the motion pattern generator translates the preprocessed target gait velocity and the current gait phase to abstract leg angles using periodic, rhythmic motions, which consist of multiple motion primitives. These abstract leg angles are a representation of the leg pose, consisting of the leg extension, three leg angles and two foot angles, and is, in contrast to representations in cartesian or joint space, designed for easy use in walking. Thus, the initial gait pose and essential walking motion primitives are represented in the abstract space. These representations are translated into the inverse space, where further motion primitives can be applied. In the end, the leg interface translates these representations of the leg pose into joint angles. The resulting gait shows satisfactory results when combined with a compliant joint setting, while still being a model-free open-loop solution.

Moreover, to further improve the performance, this method has been extended by Allgeuer et al.}[3]. The proposed approach closes the open-loop by implementing feedback mechanisms based on the Fused Angle representation[2]. Using the deviations d_ϕ and d_θ of the fused roll ϕ_B and fused pitch θ_B from the expected orientation, the model calculates the activation value of different corrective actions. These corrective actions are motion primitives, which are weighted with their activation values and then applied on the primitives of the open-loop gait. To obtain these activation values d_ϕ and d_θ are passed through a feedback pipeline (see Fig. 3.1) to produce a PID vector $\mathbf{e} \in \mathbb{R}^6$. This vector is then multiplied by a gain matrix $\mathbf{K_a} \in \mathbb{R}^{5\times 6}$ to generate the activation values. The five corrective actions we will take into account are (also see Fig. 3.1):[1]

- Arm Angle: A swinging motion of the arms is used to shift the CoM and

[1]Note however that we will explicitly not discuss the Virtual Slope corrective action[3], since it is not processed by $\mathbf{K_a}$.

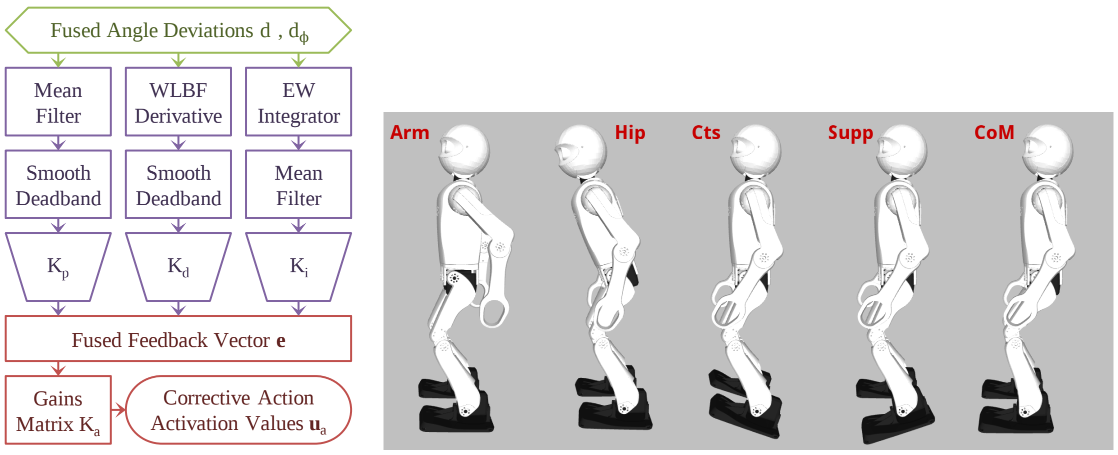

Figure 3.1: The feedback pipeline restricted to the used mechanisms (left) and a visualization of the Fused Feedback corrective actions in sagittal direction (right). **Note:** Adapted from [3].

apply a corrective impulse to the torso.

- Hip Angle: Using adjustments of the hip joints, the robot leans the torso in sagittal and lateral direction.
- Continuous Foot Angle: This parameter resembles an offset of the foot joint angles to achieve a tilt of the whole robot.
- Support Foot Angle: In contrast to the Continuous Foot Angle, this parameter depends on the gait phase and only activates the corresponding support foot. Only one foot can be active at a time by fading them in and out linearly.
- CoM Shifting: The relative position of the feet to the torso gets adjusted. This adjustment shifts the CoM, which has a direct effect on the gait.

Experiments have shown that the resulting gait is portable to different robots, sometimes even without modification of the gain matrix $\mathbf{K_a}$[3]. Nevertheless, by proper tuning of the parameters for the corresponding robot, it is possible to achieve an effective improvement of the gait. Although experience has shown that only around 10 non-zero values of these originally 30 matrix entries suffice for a satisfactory gait, still the process of manual tuning remains time consuming and requires expert knowledge over the system.

Chapter 4

Bayesian Optimization

When confronted with problems that are expensive to evaluate, Bayesian Optimization is a state-of-the-art method to find an optimum with a small amount of evaluations. With its roots in experimental design it has already shown its sample efficiency in many applications[13]. In this chapter we will lay its basis by first discussing Linear Bayesian Regression before we extend it using the Kernel Trick. Then, we will define acquisition functions which are oriented by sample efficiency. Namely, using a criterion based on the Kullback-Leibler-Divergence we will discuss the method of Entropy Search. Moreover, we will also discuss Multi-fidelity Entropy Search, a method supporting multiple sources of information.

4.1 Linear Bayesian Regression

For introducing Bayesian Optimization, we will first take a look at the standard linear model, like proposed in [24]. The general setting is, provided a countable point set $\mathbf{X} \subset \mathbb{R}^d$ of n samples with corresponding target values $f(\mathbf{X}) \subset \mathbb{R}$, we want to find a line parametrization $\boldsymbol{\omega} \in \mathbb{R}^d$ that interpolates the data best, given some prior knowledge, a *model*. Furthermore, we assume our target function $f(\mathbf{X})$ is corrupted by normal distributed noise ε with standard deviation σ. More precisely, for a sample $\mathbf{x_i} \in \mathbf{X}$ we now are looking for a function

$$f(\mathbf{x_i}) = \boldsymbol{\omega}^T \mathbf{x_i}\,, \quad i = 1, \ldots, n\,, \tag{4.1}$$

while for each x_i we can only observe

$$y_i = f(\mathbf{x_i}) + \varepsilon \tag{4.2}$$

$$\varepsilon \sim \mathcal{N}(0, \sigma^2)\,. \tag{4.3}$$

Using Eq. (4.1) and Eq. (4.2), it is possible to formulate the likelihood of the observations $\mathbf{y}$ as

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\omega}) = \prod_{i=1}^{n} p(y_i|\mathbf{x_i}, \boldsymbol{\omega})\,, \tag{4.4}$$

since we assume independence between our samples. If we further expand the product and factorize the exponential parts of the likelihood - and the scalar part respectively - we can construct a joint distribution[24]

$$\begin{aligned} p(\mathbf{y}|\mathbf{X},\boldsymbol{\omega}) &= \prod_{i=1}^{n} \left(\frac{1}{\sqrt{2\pi}\sigma} exp \left(-\frac{\left(y_i - \mathbf{x_i}^T\boldsymbol{\omega}\right)^2}{2\sigma^2} \right) \right) \\ &= \frac{1}{\sqrt{2\pi}^n \sigma^n} exp \left(-\frac{\|\mathbf{y} - \mathbf{X}^T\boldsymbol{\omega}\|_2^2}{2\sigma^2} \right) . \end{aligned} \tag{4.5}$$

Thus, we get a distribution $p(\mathbf{y}|\mathbf{X},\boldsymbol{\omega}) = \mathcal{N}(\mathbf{X}^T\boldsymbol{\omega}, \sigma^2 Id)$, which can be used to create a posterior distribution $p(\boldsymbol{\omega}|\mathbf{y},\mathbf{X})$ using a prior model $p(\boldsymbol{\omega})$ by applying the Bayes rule[24]

$$p(\boldsymbol{\omega}|\mathbf{X},\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{X},\boldsymbol{\omega})p(\boldsymbol{\omega})}{p(\mathbf{y}|\mathbf{X})} . \tag{4.6}$$

Please note that the marginal likelihood $p(\mathbf{y}|\mathbf{X})$ does not depend on $\boldsymbol{\omega}$ and can thus be neglected as a normalization factor over all $\boldsymbol{\omega}$. Additionally, we assume our prior model to be zero-mean Gaussian distributed with covariance matrix $\boldsymbol{\Sigma}$

$$\boldsymbol{\omega} \sim \mathcal{N}(0, \boldsymbol{\Sigma}) . \tag{4.7}$$

Neglecting the marginal likelihood, after the application of the Bayes formula and the matrix inversion lemma we get the resulting posterior[24]

$$\begin{aligned} p(\boldsymbol{\omega}|\mathbf{X},\mathbf{y}) &\propto \exp\left(-\frac{1}{2\sigma^2} \left(\mathbf{y} - \mathbf{X}^T\boldsymbol{\omega}\right)^T \left(\mathbf{y} - \mathbf{X}^T\boldsymbol{\omega}\right) \right) \exp\left(-\frac{1}{2}\boldsymbol{\omega}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\omega} \right) \\ &\propto \exp\left(-\frac{1}{2} \left(\boldsymbol{\omega} - \bar{\boldsymbol{\omega}}\right)^T \left(\frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T\boldsymbol{\Sigma}^{-1} \right) \left(\boldsymbol{\omega} - \bar{\boldsymbol{\omega}}\right) \right) \end{aligned} \tag{4.8}$$

with $\bar{\boldsymbol{\omega}} = \frac{1}{\sigma^2}\left(\frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T + \boldsymbol{\Sigma}^{-1}\right)^{-1}\mathbf{X}\mathbf{y}$. By introducing $\mathbf{A} = \frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T + \boldsymbol{\Sigma}^{-1}$, it is possible to build a posterior Gaussian distribution

$$p\left(\boldsymbol{\omega}|\mathbf{X},\mathbf{y}\right) \sim \mathcal{N}\left(\bar{\boldsymbol{\omega}}, \mathbf{A}^{-1}\right) , \tag{4.9}$$

whose mean $\bar{\boldsymbol{\omega}}$ is the *maximum-a-posteriori* (MAP) estimate[24]. The MAP estimate here is the most probable instance of $\boldsymbol{\omega}$ with our provided data and model.

It is furthermore important to see that $p(\mathbf{y}|\mathbf{X},\boldsymbol{\omega})$ defines a probability distribution for each point in the input space, thus resembling a stochastic process over $\mathbf{X}$. Since we previously assumed normal distributed noise over our target space (likelihood), the resulting stochastic process is called *Gaussian Process* (GP).

To receive an estimate f_* for a specific point $\mathbf{x}_*$, it is necessary to take all possible models into account and multiply them by their likelihood. Thus, if we restrict our estimate to $\mathbf{x}_*$, we will receive

$$\begin{aligned} p(f_*|\mathbf{x}_*,\mathbf{X},\mathbf{y}) &= \int p(f_*|\mathbf{x}_*,\boldsymbol{\omega})p(\boldsymbol{\omega}|\mathbf{X},\mathbf{y})d\boldsymbol{\omega} \\ &= \mathcal{N}\left(\mathbf{x}_*^T\bar{\boldsymbol{\omega}}, \mathbf{x}_*^T\mathbf{A}^{-1}\mathbf{x}_*\right) \end{aligned} \tag{4.10}$$

by applying $\mathbf{x}_*$ to Eq. (4.9).

The approach of Bayesian Linear Regression can be extended using the *Kernel Trick*, the basis of all Kernel Methods, so that non-linear functions can be learned. This key idea will lead us to the topic of non-linear Bayesian regression.

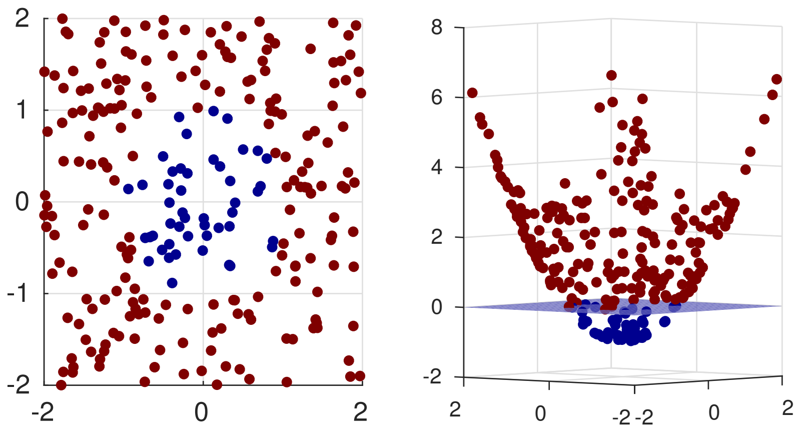

Figure 4.1: An example of linear separability in a higher dimensional feature space. The red and blue dots of the input data are not linearly separable in the input space (left), but if transformed into a higher dimensional feature space using a non-linear transformation $\phi(x, y, z) = (x, y, x^2 + y^2 - 1)^T$ it becomes linearly separable (right).

4.2 Non-Linear Bayesian Regression

Applying the Kernel Trick to Linear Bayesian Regression, it is possible to transform the input space into a feature space. This feature space is typically of much higher dimension to enable linear regression methods to learn non-linear functions according to Cover's theorem[6]. An illustration of the concept of Cover's theorem applied on a classification problem can be found in Fig. 4.1. If we want to implement this transformation explicitly, we will potentially need to perform all calculations in the feature space, including expensive matrix inversions[24]. Fortunately, it is possible to reduce all calculations to pairwise inner products of the samples and we can define this inner product without dealing with the feature space explicitly.

Namely, we extend Eq. (4.10) by a Kernel function ϕ for a vector input and $\mathbf{\Phi} = \phi(\mathbf{X})$ as the application of ϕ on the data set $\mathbf{X}$ to receive a distribution

$$p(f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}\left(\frac{1}{\sigma^2}\phi(\mathbf{x}_*)^T \mathbf{A}_\phi^{-1}\mathbf{\Phi}\mathbf{y}, \phi(\mathbf{x}_*)^T \mathbf{A}_\phi^{-1}\phi(\mathbf{x}_*)\right) \tag{4.11}$$

in the feature space spanned by ϕ for $\mathbf{A}_\phi = \frac{1}{\sigma^2}\mathbf{\Phi}\mathbf{\Phi}^T + \mathbf{\Sigma}^{-1}$. Since Eq. (4.11) would invoke an expensive inversion of $\mathbf{A}_\phi$ (recall $\mathbf{\Phi}$ is high dimensional), we expand $\mathbf{A}_\phi$ and rewrite the distribution

$$p(f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\phi(\mathbf{x}_*)^T \mathbf{\Sigma}\mathbf{\Phi}\left(\mathbf{K} + \sigma^2 Id\right)^{-1}\mathbf{y}, \tag{4.12}$$

$$\phi(\mathbf{x}_*)^T \mathbf{\Sigma}\phi(\mathbf{x}_*) - \phi(\mathbf{x}_*)^T \mathbf{\Sigma}\mathbf{\Phi}\left(\mathbf{K} + \sigma^2 Id\right)^{-1}\mathbf{\Phi}^T\mathbf{\Sigma}\phi(\mathbf{x}_*)) \tag{4.13}$$

for $\mathbf{K} = \mathbf{\Phi}^T\mathbf{\Sigma}\mathbf{\Phi}$, requiring multiple steps which can be found in [24]. It can be observed, that in Eq. (4.13) all matrix inversions are in the input space, thus in a typically much lower dimension compared to Eq (4.11). This way, it is possible to benefit from the power of the feature space, without sacrificing too much computational power.

Note also that it is not even necessary to know the shape of the corresponding feature space, since we will only work with the inner products in the input space. The function that defines these inner products is called the *Kernel*.

In this manner, we need to find and to parametrize a suitable kernel function in order to learn non-linear functions. Literature offers a large selection of kernel functions for different purposes, where two of the most common kernels are the squared exponential (SE) kernel

$$k_{SE}(\mathbf{x},\mathbf{x}') = \sigma^2 \exp\left(-\frac{1}{2}\left(\mathbf{x}-\mathbf{x}'\right)\boldsymbol{l}^{-1}\left(\mathbf{x}-\mathbf{x}'\right)\right) \tag{4.14}$$

and the rational quadratic (RQ) kernel

$$k_{RQ}(\mathbf{x},\mathbf{x}') = \sigma^2\left(1+\frac{1}{2\alpha}\left(\mathbf{x}-\mathbf{x}'\right)\boldsymbol{l}^{-1}\left(\mathbf{x}-\mathbf{x}'\right)\right)^{-\alpha}, \tag{4.15}$$

where the squared exponential kernel assumes very smooth and even functions, while the rational quadratic kernel is not as strict in that regard[13]. Note that both kernels have specific parameters that need to be tuned, typically called hyperparameters. Both kernels share the specific lengthscale $\boldsymbol{l} \in \mathbb{R}^d$ which roughly determines the distance of two points to majorly influence each other, while a scale factor $\sigma^2 \in \mathbb{R}$ determines the problem-specific signal variance. $\alpha \in \mathbb{R}$ on the other hand is only used by the RQ kernel and is a relative weight scale for the variation of the lengthscale[24].

In this manner, if a suitable kernel function is applied, the resulting function estimate can be used to create an estimate over the location of its minimum, namely by minimizing the posterior mean function. Using the variance of this point, it is possible to evaluate the certainty of that candidate being the actual minimum. Thus, we can use the Gaussian Process framework as an optimization tool.

On the other hand, if we want to perform *active learning*, we need to address the problem of choosing the evaluation points at each iteration. Prominent examples of those acquisition functions are *probability of improvement* and *expected improvement*. For a parameter interval $X \subseteq \mathbb{R}^d$ and a target function $J\colon X \to \mathbb{R}$, the former will choose a point $\boldsymbol{\theta} \in X$ that has the highest probability to be better than the current best guess $\boldsymbol{\theta_{bg}}$, thus optimizing

$$\boldsymbol{\theta_{t+1}} = \operatorname*{arg\,max}_{\boldsymbol{\theta}\in X}\; p(\mathbb{E}[J(\boldsymbol{\theta})] < \mathbb{E}[J(\boldsymbol{\theta_{bg}})])\,. \tag{4.16}$$

In contrast, the latter takes into account the expected difference of the function values by choosing

$$\boldsymbol{\theta_{t+1}} = \operatorname*{arg\,max}_{\boldsymbol{\theta}\in X}\; \max(\mathbb{E}[J(\boldsymbol{\theta})] - \mathbb{E}[J(\boldsymbol{\theta_{bg}})], 0)\,. \tag{4.17}$$

A further extension of these acquisition functions can be found in the next sections.

4.3 Entropy Search

With the aim of globally optimizing a function $J(\boldsymbol{\theta})$, Entropy Search (ES) uses ideas from experimental design to reduce the amount of necessary evaluations[13]. The belief p_{min} over the location of the minimum is approximated by a non-uniform grid, which, upon convergence, will be peaked around the actual minimum[13]. Using a Gaussian Process (see Section 4.1) it is possible to model noise on the input data while approximating the target function efficiently. Moreover ES uses the following acquisition function

$$\boldsymbol{\theta_{t+1}} = \underset{\boldsymbol{\theta} \in X}{\arg\max} \left(\mathbb{E}(\Delta H(\boldsymbol{\theta}))\right) \tag{4.18}$$

based on the expected change of entropy ΔH, such that the most informative point is efficiently chosen at each iteration. There are multiple approximations needed to perform these calculations, which can be found in [13].

By using the relative entropy, ES is able to locate the minimum with only a fraction of the evaluations that were necessary for comparable optimization algorithms - a property especially useful for problems with expensive function evaluations[13].

However, it is important to note that the performance of the algorithm depends on the prior assumptions used in the GP. If the GP prior represents the actual data noise and covariance only poorly, this may lead to unwanted behavior of the algorithm. Small discrepancy however is typically caught by the probabilistic nature of the method.

4.4 Multiple information sources

Whilst in the preceding sections we only discussed data that originated from one source, [18] extended the ES algorithm to integrate multiple sources of information. The resulting method is called *Multi-fidelity Entropy Search* (MF-ES) and typically trades off real experiments with simulations. MF-ES optimizes the following cost function

$$J_{real}(\boldsymbol{\theta}) = J_{sim}(\boldsymbol{\theta}) + \varepsilon_{sim}(\boldsymbol{\theta}) \tag{4.19}$$

over a parameter set $\boldsymbol{\theta} \in X$, which in turn now is composed of two costs, J_{sim} and ε_{sim}. The measured cost on the physical system is denoted as J_{real} and is, if not measured directly, expressed as the measured cost in the simulation J_{sim} and an error term $\varepsilon_{sim}(\boldsymbol{\theta})$. It is important to understand that the systematic simulator error $\epsilon_{sim}(\boldsymbol{\theta})$ can be a complex transformation and is inevitably required to infer the real cost from J_{sim}. However, this transformation can be learned directly by incorporating ϵ_{sim} into the Gaussian Process, which is the key idea of MF-ES.

Consequently, the approach introduces two kernel functions k_{sim} and k_{ε}, where the latter resembles the systematic simulator noise. Furthermore, to compare simulator evaluations with real experiments, it is required to introduce a binary flag δ, where $\delta = 1$ indicates a real experiment and $\delta = 0$ a simulation respectively[18]. Thus, we conclude with a kernel function

$$k(\mathbf{a}, \mathbf{a}') = k_{sim}(\boldsymbol{\theta}, \boldsymbol{\theta'}) + k_{\delta}(\delta, \delta')k_{\varepsilon}(\boldsymbol{\theta}, \boldsymbol{\theta'})\,, \tag{4.20}$$

with a parameter extension $a = (\delta, \boldsymbol{\theta})$ and a kernel indicator $k_\delta(\delta, \delta') = \delta\delta'$, which is only active if both evaluations were performed in the real world[18]. Accordingly, two real experiments are expected to covary stronger than the cases where one evaluation has been performed in simulation, i.e. where the covariance is solely expressed through k_{sim}.

Again, it is useful to highlight the consequence of the introduction of k_ε, which encodes the transformation between the simulated and the real world. Since k_ε is modeled inside the GP, it is not necessary to address possible transformations between J_{sim} and J_{exp} explicitly - it only requires assumptions about the shape of the error in form of a mean and a covariance function.

It is important to note, however, that δ has to be explicitly incorporated into the acquisition function of the algorithm, since the algorithm would otherwise only choose to evaluate real experiments, which naturally deliver most information about the target function. To address this issue, trade-off parameters are introduced for both information sources. More precisely, the ES acquisition function is extended by positive weights u_i resulting in the following selection criterion

$$\boldsymbol{\theta_{t+1}} = \underset{\boldsymbol{\theta}\in\mathbb{R}^d, i\in\{sim,real\}}{\arg\max} \left(\frac{\Delta H_t(\boldsymbol{\theta})}{u_i} \right). \tag{4.21}$$

The application of this method on a synthetic example is visualized in Fig. 4.2.

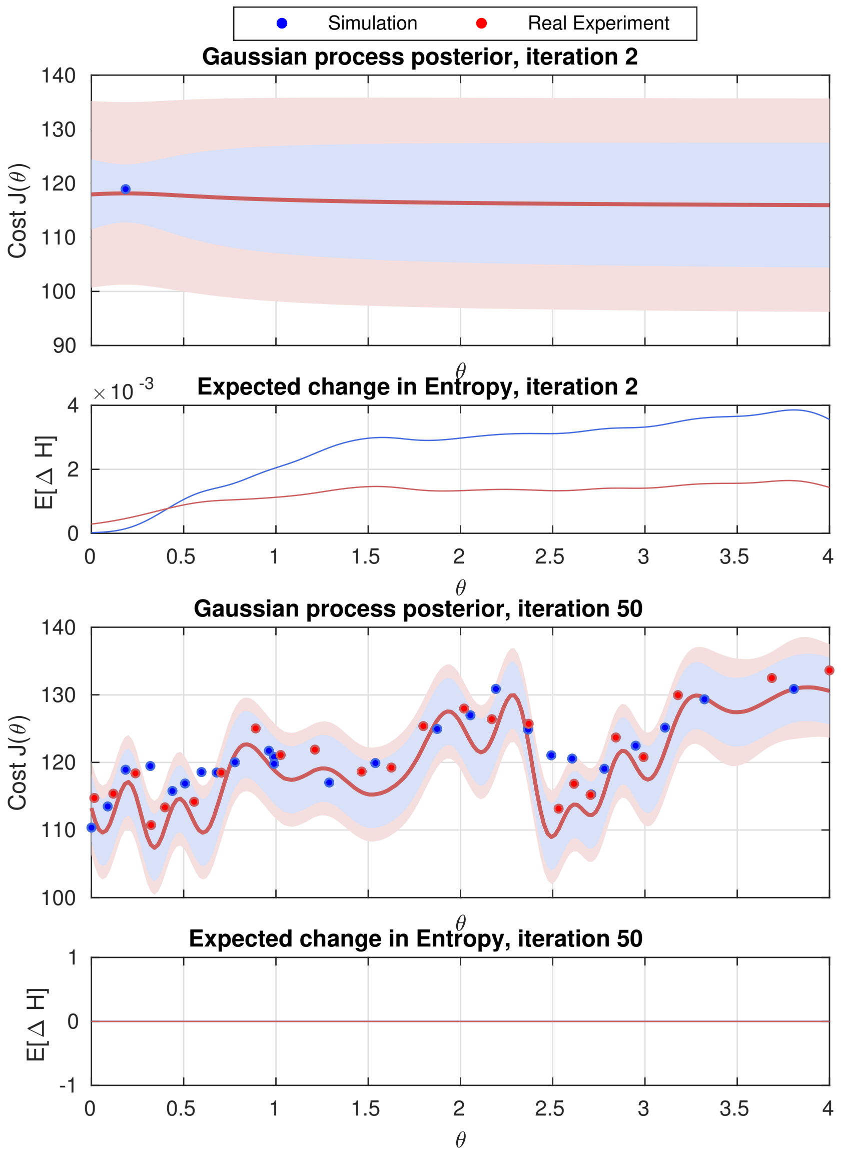

Figure 4.2: A synthetic example for a 1D optimization using Multi-fidelity Entropy Search, where real experiments were substituted by simulations. In the GP visualizations, simulations are shown as blue dots, real experiments (simulated) in red. The surfaces in the corresponding colors resemble the covariance of the simulation and the real experiments. Below the GP visualizations the corresponding expected change in entropy is depicted, which is remarkably higher in early iterations (upper figures).

Chapter 5

Sample-efficient Gait Learning using MF-ES

In this chapter we introduce a framework that utilizes Multi-fidelity Entropy Search to learn the Fused Feedback gains of the NimbRo gait. This includes the design of a cost function, the search of suitable hyperparameters for a GP model and a proposed termination criterion.

5.1 Cost function

A preliminary for the application of Reinforcement Learning approaches is the design of a cost function. Since the system learns exactly the models that minimize this cost function, it is important that it reflects the actual goal of the optimization. In the scope of this thesis, the main goal of the optimization is an improvement of the walking stability, while keeping the required amount of iterations low. The final cost function will include parts of the Fused Feedback as a stability measure, since a gait with a low activation of corrective actions is favorable. Furthermore, the evaluations of simulations are computed by the mean of multiple trials to reduce the simulator noise. Although reduced noise of real-world experiments is also beneficial, the proposed measure would be too expensive and thus is only used for simulations. Finally, we will introduce a regularization that prioritizes small parameters over parameters of large magnitude, since latter typically result in higher torques.

In this manner, we first have to address the problem that simulators can be noisy. However, since simulations are generally cheap to perform, the easiest approach is to work with the mean of $N > 0$ evaluations

$$\bar{J}_{sim}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} J_{sim}(\boldsymbol{\theta}) \,, \tag{5.1}$$

for a parameter $\boldsymbol{\theta} \in X$ inside a compact interval $X \subset \mathbb{R}^d$. Of course, if confronted with outliers of high magnitude, it is also possible to use the median instead of the mean. Further reduction of noise can be achieved by careful construction of the cost function J.

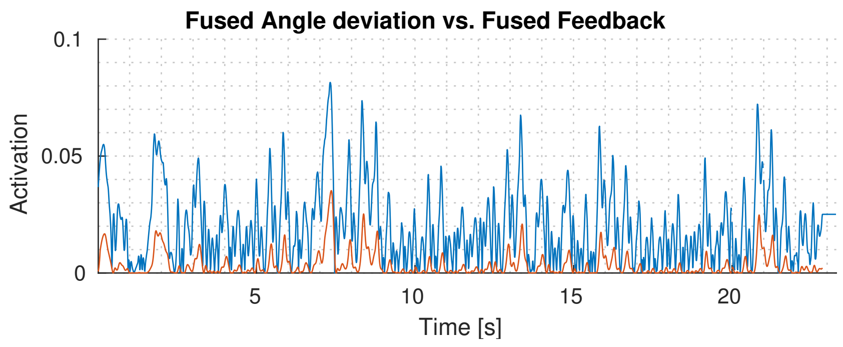

Figure 5.1: This figure displays the difference between the absolute Fused Angle deviations $\|d_\alpha\|_1$ (blue) and the absolute proportional Fused Feedback $\|e_{P,\alpha}\|_1$ (red). Due to the deadband and mean filter, which are applied to the Fused Angle deviations to yield the Feedback values, the Fused Feedback signal is smoother and low amplitudes get rejected.

Generally, since the Fused Angle deviations[1] d_α(pitch) and d_β(roll) represent the unintended tilt that is induced by walking, it can be used as a measure of the gait stability, with

$$d_\alpha = \alpha_{expected} - \alpha_{measured} \tag{5.2}$$

$$d_\beta = \beta_{expected} - \beta_{measured} \,. \tag{5.3}$$

Thus, we want our cost function to be proportional to these measurements to achieve optimized parameters in terms of gait stability. The most direct way to accomplish this proportionality is to integrate over the deviations during the experiments of duration T

$$J(\boldsymbol{\theta}) = \int_0^T \|d_\alpha(\boldsymbol{\theta})\|_1 + \|d_\beta(\boldsymbol{\theta})\|_1 dt \,. \tag{5.4}$$

Note that since measurements are taken at discrete time steps, the integral in Eq. (5.4) is in fact realized through a sum. Although this is a straight forward definition of the cost, it comes with downsides, for example noise sensitivity. Due to the fact that the measurements are taken in the control loop at a high rate, measurement errors can easily sum up and distort the result. To cope with that problem, we apply a mean filter and a smooth deadband to the measurements like proposed in [3]. So, we only consider the proportional part $(e_{P\alpha}, e_{P\beta})$ of the Fused Feedback vector $\mathbf{e} = (e_{P\alpha}, e_{P\beta}, e_{I\alpha}, e_{I\beta}, e_{D\alpha}, e_{D\beta})^T$ and consequently modify the cost function as

$$J(\boldsymbol{\theta}) = \int_0^T \|e_{P\alpha}(\boldsymbol{\theta})\|_1 + \|e_{P\beta}(\boldsymbol{\theta})\|_1 dt \,, \tag{5.5}$$

which still represents the notion of the Fused Angle deviations, like shown in Fig. 5.1.

Furthermore, this modification can also be thought of removing the necessity for the algorithm to learn the different filter elements of the feedback pipeline.

[1]To prevent confusion about the notation and contrast it to the parameters $\boldsymbol{\theta}$, the Fused Angle pitch is denoted as α and the roll as β respectively in contrast to the notation in Chapter 3.

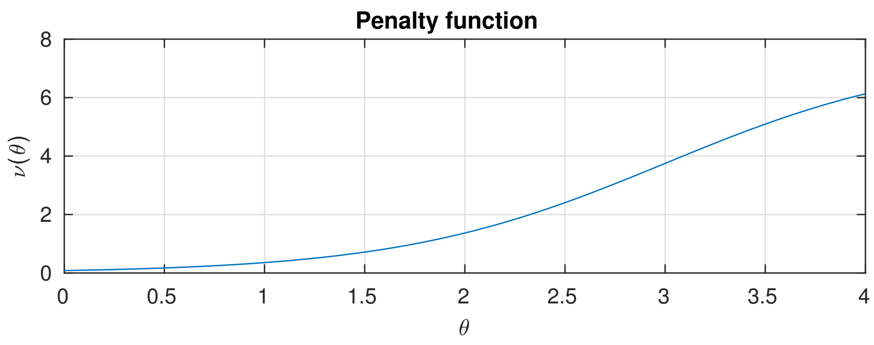

Figure 5.2: $\nu(\boldsymbol{\theta})$ uses the logistic function to create a smooth penalty function against parameters of large magnitude. Here, we use $\boldsymbol{\theta_{max}} = 4$, $s = 7.5$, $\lambda = 0.75$ and $\gamma = 6$.

Moreover, it is favorable to regularize the magnitude parameters, since higher values typically result in higher servo torques. Thus, we define a loss function $\nu\colon X \to \mathbb{R}$, which smoothly penalizes parameters of higher magnitude. Generally, we regularize using a logistic function with maximum gradient where $\|\boldsymbol{\theta}\|_2 = \lambda\boldsymbol{\theta_{max}}$ for $\lambda \in [0, 1]$

$$\nu(\boldsymbol{\theta}) = \frac{s}{1 + \exp\left(-\gamma(\|\boldsymbol{\theta}\|_2 - \lambda\|\boldsymbol{\theta_{max}}\|_2)\right)}\,. \tag{5.6}$$

Where $\boldsymbol{\theta_{max}}$ is the maximum value of our current parameter $\boldsymbol{\theta}$, the penalty magnitude $s \in \mathbb{R}$ has to be defined depending on how hard large parameters shall get penalized and $\gamma \in \mathbb{R}$ controls the smoothness of the phase transition. An example setup of this penalty function can be seen in Fig. 5.2.

Moreover, we restrict the optimization to parameters in a common plane (i.e. sagittal & lateral). Since corrective actions in the sagittal plane are only activated by the Fused Angle pitch α and in the lateral plane by the Fused Angle roll β, their performance is sufficiently expressed in the respective plane. Restricting the cost function to these planes further reduces the impact of noise, which leads to a better performance of our optimization. In this manner we propose two final cost functions

$$J_{exp,\alpha}(\boldsymbol{\theta}) = \int_0^T \|e_{P\alpha}(\boldsymbol{\theta})\|_1 dt + \nu(\boldsymbol{\theta}) \tag{5.7}$$

$$J_{exp,\beta}(\boldsymbol{\theta}) = \int_0^T \|e_{P\beta}(\boldsymbol{\theta})\|_1 dt + \nu(\boldsymbol{\theta})\,, \tag{5.8}$$

to evaluate the gait of corrective actions in the two different planes separately.

For the real world experiments we remain with a cost function J_{exp} equal to Eq. (5.7) and Eq. (5.8), while for the simulations the final cost functions are expressed as

$$\bar{J}_{sim,\alpha}(\boldsymbol{\theta}) = \frac{1}{N}\left(\sum_{i=1}^{N}\int_0^T \|e_{P\alpha}(\boldsymbol{\theta})\|_1 dt\right) + \nu(\boldsymbol{\theta}) \tag{5.9}$$

$$\bar{J}_{sim,\beta}(\boldsymbol{\theta}) = \frac{1}{N}\left(\sum_{i=1}^{N}\int_0^T \|e_{P\beta}(\boldsymbol{\theta})\|_1 dt\right) + \nu(\boldsymbol{\theta})\,. \tag{5.10}$$

In order to improve the performance of the learning algorithm, the cost function above deals with the magnitude of distortion by simulator noise.

Furthermore, it is necessary to address cases where the robot falls during an experiment. Since the measured deviations during these falls can exceed the expressibility of the kernel function, they would bias surrounding parameters towards a high cost. In order to limit this negative effect, we introduce a maximum cost

$$J_{i,max} = \mu_i + \psi_J \,, \tag{5.11}$$

where $\psi_J \in \mathbb{R}$ resembles the admissible distance of a measured cost to the prior mean $\mu_i \in \mathbb{R}$ for $i \in \{sim, exp\}$. This maximum cost is then used as an upper bound for the cost

$$\hat{J}_i(\boldsymbol{\theta}) = \min\left(J_i(\boldsymbol{\theta}), J_{i,max}\right) \,. \tag{5.12}$$

5.2 Hyperparameters

Since it is not possible to describe every Gaussian Process with the same parameters, it is necessary to find a suitable model and its respective parameters like explained in Section 4. We decided to use the rational quadratic (RQ) kernel, since it is not as smooth as the squared exponential (SE) kernel, and optimized the parameters for use in a 2D optimization. In this application we prefer the less smooth RQ kernel, since we assume the problem of gait learning to be of non-smooth nature. For the 4D optimization, we deduce the values from the 2D experiments.

The choice of the RQ kernel requires to determine 3 parameters, namely a lengthscale l, a signal variance s^2 and an exponent α, but it also leaves a prior mean and covariance to be set. While l and α are parameters that have to be chosen by the user based on the expected shape on the function, an approximation of s^2 can be found empirically. Note, however, that with enough data, l and α can be inferred. Since we want to keep the amount of prior data low, we manually chose a set of parameters that promised to trade off exploration and exploitation adequately.

In this manner, we chose $\boldsymbol{l} = \left(\frac{\boldsymbol{\theta}_{i_{max}}}{8}\right)_i \in \mathbb{R}^d$ and $\alpha = 0.25$ to produce a reasonable trade-off between exploration and exploitation, where $\boldsymbol{\theta}_{i_{max}}$ is the maximum value of parameter $\boldsymbol{\theta}_i$. We use the same lengthscales for the simulation as for the real evaluations. Additionally, we suggested the prior standard deviation to be $s_{tot} = 10$ to model the initial uncertainty of the model, while we propose simulations reduce this uncertainty for the real model by $\kappa = 0.6$. The standard deviations are then calculated as

$$\begin{aligned} s_{sim} &= \kappa\, s_{tot} = 6 \\ s_{real} &= s_{tot}\sqrt{(1-\kappa^2)} = 8 \,. \end{aligned} \tag{5.13}$$

Moreover, using 5 evaluations of the real robot and simulation each, values for the standard deviations were estimated. Undoubtedly, this small number of evaluations will not result in an exact estimate, but the resulting parametrization still was good enough to produce satisfactory results in our actual application. The standard deviation of the evaluations in the simulator was set to $\sigma_{sim} = 0.1298$ and the standard deviation of the difference between a simulation and a real experiment was set to $\sigma_\varepsilon = 5.3515$.

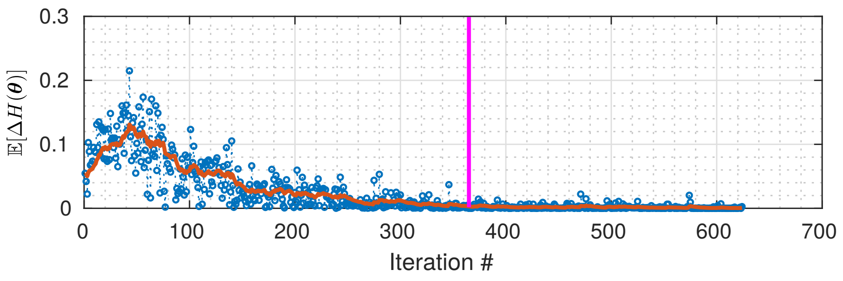

Figure 5.3: The figure shows the evolution of the Kullback-Leibler-Divergence in a simulation-only run. The blue graph shows the unfiltered KLD over the iterations, whereas the orange data has been filtered by a saturated filter with $v = 0.9$. The vertical magenta line displays the iteration, where a threshold of $\varepsilon_{KLD} = 0.003$ would have stopped the learning process.

Furthermore, for the sake of visualization, we ensure that the evaluated points are close to the mean function by setting the corresponding means $\mu_{sim} = 53.3502$ and $\mu_\varepsilon = -37.1385$. This also positively effects the estimate around the domain borders and thus improves the performance of the algorithm especially in early iterations.

5.3 Termination criteria

In the context of global optimization it is very important to determine when to stop the algorithm. The most simple and frequently used criteria are based on the number of iterations. Whereas this condition works fine for problems that are fast to compute, it looses applicability when iterations become more expensive, especially if there is no prior knowledge over the behavior of the algorithm on a given dataset. Performance driven conditions are often based on the predictions of a test set and stop the algorithm when the performance on the test increases slow enough, or even decreases in case of overfitting. In any case, these criteria require a test set, which is normally constructed as subset of the dataset for large scale data. This is not feasible when the data is highly limited, and so it is favorable to use any available data in the learning procedure. Using only a small test set, i.e. a hand full of reference points, will not be representative to the problem and may result in unwanted behavior.

In this manner, we propose a termination criterion which is originated from the idea of Entropy Search[13]. The *Kullback-Leibler-Divergence*[2] (KLD) between the prior and expected posterior of an iteration expresses the expected amount of gained information that results by evaluating the current query point. We formulate a criterion that stops the algorithm as soon as the expected Kullback-Leibler-Divergence $\mathbb{E}(\Delta H(\boldsymbol{\theta}_t))$ is no longer significant. However, to ensure that outliers do not lead to a premature stop, we apply a saturated filter (Fig. 5.3) defined for the iteration t and a velocity factor $0 < v < 1$ as follows

$$KLD_1 = \mathbb{E}(\Delta H(\boldsymbol{\theta}_1)) \tag{5.14}$$

$$KLD_t = (1 - v)KLD_{t-1} + v\mathbb{E}(\Delta H(\boldsymbol{\theta}_t)) \tag{5.15}$$

[2]Also known as relative Entropy or Information Gain.

Consequently, we stop the algorithm if $KLD_t < \varepsilon_{KLD}$ for a *KLD-threshold* $\varepsilon_{KLD} > 0$. Since the filter is not saturated in the beginning, we propose a minimum number of iterations that are performed before the KLD-threshold criterion is applied. This is necessary, since a bad prior mean can lead to a low KLD right after the first iteration and thus would lead to a premature stop. We propose a KLD-threshold $\varepsilon_{KLD} = 0.01$, which would have stopped a synthetic 2D optimization after 261 iterations (see Fig. 5.3). Moreover, in Fig. 5.3 you can see the impact of the proposed filter. While the raw data is fluctuating and would have lead to a fast termination of the algorithm, the filtered data, however, is more stable and can be used by the criterion.

Nonetheless, it is favorable to use a composition of termination criteria, since the convergence of the KLD cannot be guaranteed if our prior assumptions do not represent reality well-enough. Thus, we use the KLD-threshold criterion in combination with a maximum number of iterations to set a hard time constraint.

Chapter 6

Setup & Evaluation

In this chapter, we will discuss the application of our approach on the igus® Humanoid Open Platform. Moreover, we will evaluate the results and highlight the performance of our method.

6.1 Experiment Setup

As a robot platform we chose to use the igus® Humanoid Open Platform[4] (see Fig. 6.13), which is 92cm tall and weights 6.6kg. It has 20 Degrees of Freedom and is equipped with a gyroscope and an accelerometer. The platform has already shown its robustness during multiple RoboCup competitions[11]. Furthermore, the robot uses a Gigabyte BRIX computer with an Intel i7-5500U and 4GB of RAM. The igus® Humanoid Open Platform is equipped with 6 Robotis MX-106 actuators per leg and 8 MX-64 for the remaining joints.

Due to the robots low computational power, this learning approach has been distributed over two systems (see Fig. 6.2). The optimization and all simulations have been performed on a Desktop-PC with an 8-core Intel i7-4890K and 8GB of RAM, where the optimization has been performed in MATLAB R2017b. All used computers are set up with Ubuntu 14.04 and ROS Indigo and Gazebo 2.2.6 as a simulator (see Fig. 6.1). This PC was able to achieve a real-time factor of 1.5 in the simulator, which would not be possible on the robot, thus the process was significantly sped up.

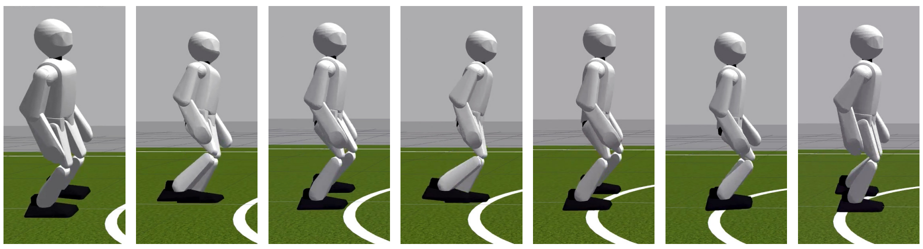
Figure 6.1: The igus® Humanoid Open Platform walking in simulation. The shown gait uses parameters resulting from a simulation-only optimization terminated by the KLD-threshold criterion.

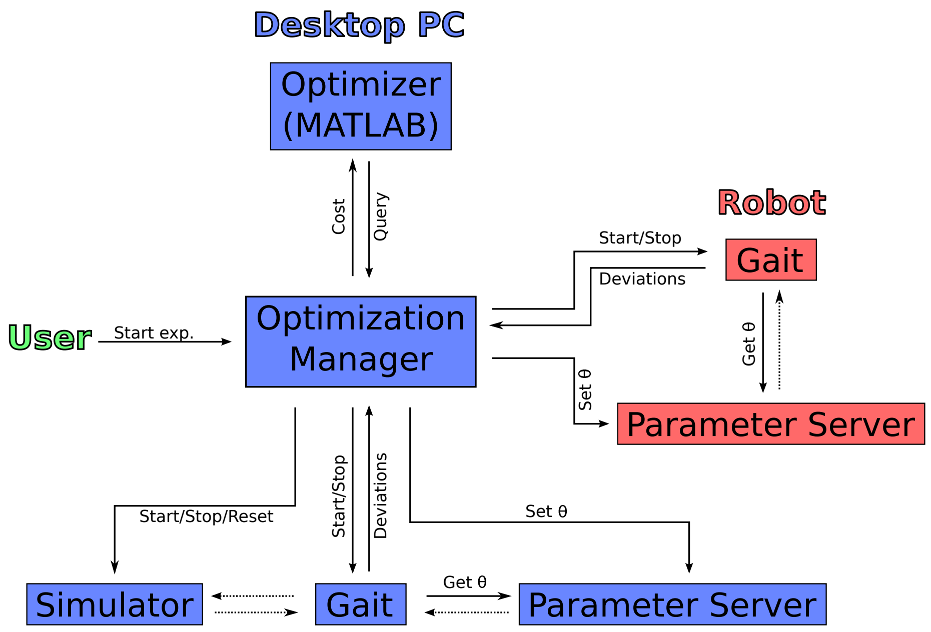

Figure 6.2: The used software architecture. The graph depicts the most important ROS nodes and the communication between them. The blue boxes indicate processes on the Desktop PC and the red boxes on the robot respectively.

Furthermore, we developed an interface between the optimizer in MATLAB and the robot, which redirects all queries and starts the corresponding experiment. Once an experiment is completed in simulation, the simulator is reset by a custom resetter to reduce simulator induced noise. Still, the problem of noise could not be sufficiently addressed at that point. Real-world experiments get forwarded to the robot over an Ethernet connection. The robot was supervised during all real experiments, thus we implemented the interface to block the start of the experiment until the supervisor has triggered the trial.

Additionally, we need to specify a parametrization for the penalty function $\nu(\theta)$. We chose $\lambda = 0.75$ and $s = 7.5$ to achieve a high level of punishment for parameters larger than $\lambda\theta_{max}$. Furthermore, the smoothness of the phase transition has been set to $\gamma = 6$. A 1D plot of this parametrized penalty function can be seen in Fig. 5.2. Moreover, in case the robot falls, we set $\psi_J = 25$ as the admissible positive deviation from the prior mean.

All performed experiments followed the same procedure: The robot was queried to walk on the spot for 3 seconds, after which it starts walking forward for 10 seconds. During all time the robot calculates its expected and measured Fused Angle, resulting in the Fused Angle deviations used in the cost function. We evaluate our approach in two different configurations. Firstly, we featured a 2D optimization restricted on only the ArmAngleY P- and D-gain, while FootAngleY and timing was enabled - all other feedback mechanisms have been disabled. The second configuration extends the previous one by using a 4D optimization of both ArmAngleY and FootAngleY P- and D-gains. Furthermore, timing and all other feedback mechanisms were disabled in this configuration to evidence the real contribution of the optimization of the selected parameters.

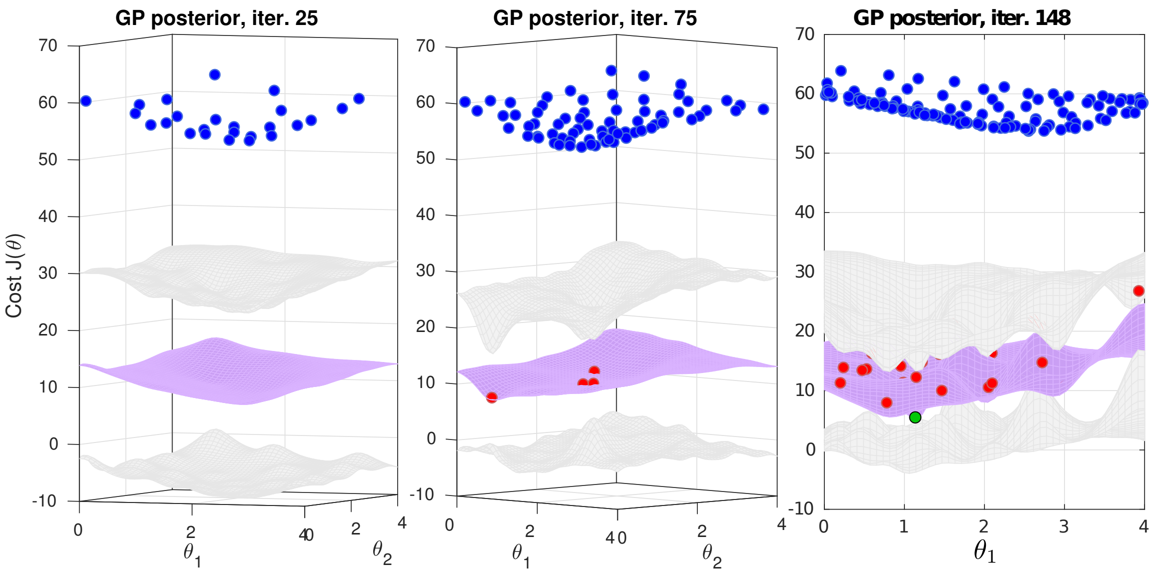

Figure 6.3: Visualization of the Gaussian Process after the 2D optimization of the ArmAngleY parameters. Real-world evaluations are shown red, whereas simulations are blue. The purple mesh resembles the posterior mean, the gray mesh shows the covariance and the green dot resembles the current minimum estimate. Due to the simulator error, simulations result in a higher cost. This transformation is learned by the GP to efficiently use simulator data in the learning process.

Additionally, we will perform a short evaluation of the KLD termination criterion using a simulator-only run of the algorithm.

6.2 Evaluation

6.2.1 2D ArmAngleY Optimization

In this configuration, the ArmAngleY proportional and derivative corrective actions and timing were enabled. The optimization was stopped after 149 iterations once the constraint of 25 real-world experiments was reached, resulting in 124 simulations. The cost of the original parameters was 16.1458, whereas the optimized parameters resulted in a cost of 16.2679. Consequently, the optimization process was able to yield parameters that are only 0.78% worse than parameters that have been manually tuned by an expert.

Still, since only ArmAngleY P- and D-gain were optimized, the process was hardly restricted to local search over all parameters that define the gait. The simultaneous optimization of more parameters is more promising and will be discussed in Sec. 6.2.3. A visualization of the resulting posterior distribution after the 2D optimization process, can be found in Fig. 6.3.

However, to depict the behavior of the algorithm, it is useful to take the development of the Gaussian Process posterior into account, like shown in Fig. 6.4. The top row of the figure displays the distribution of the points, the middle row shows the GP posterior with simulations and the bottom row without respectively. It is possible to observe that the point distribution is not uniform.

Especially in early iterations, the difference between the middle and bottom row is apparent. The simulations influence the estimate over the real experiments and thus lead the selection of the query points for the real robot. However, the more real experiments are performed, the less information of the simulator is transferred, because of the corresponding kernel hyperparameters. Nonetheless, since the real robot is the actual target function for the algorithm, it is favorable to fit tighter to the real data. It is also possible to observe, that simulator data generally corresponds to a higher cost. One reason for this is simulator induced noise, which accumulates over the trial.

6.2.2 2D FootAngleY Optimization

Using the insights of the optimization described in Sec. 6.2.1, we adapted the setup slightly and performed another optimization. In this manner, the hyperparameters of the kernel have been modified and we decided to optimize FootAngleY instead of ArmAngleY. The latter decision was lead by the experience, that the influence of the foot parameter on the gait stability was larger, especially in the simulator (see Fig 6.5).

Since it was discovered that the simulator only had small influence on the real world posterior (see Fig. 6.4), the hyperparameters of the Gaussian Process were modified. Namely, to fix $s_{exp} = 8$ and increase $s_{sim} = 12$, the corresponding variables $s_{tot} = \sqrt{208}$ and $\kappa = \frac{12}{\sqrt{208}}$ were set (see Sec. 5.2). Furthermore, the modeled noise of the real world has been corrected, such that $\sigma_{sim} = 2.32$, and small modifications have been made to the software.

The algorithm was compared with Random Search like described in Alg. 1. Random Search corrupts the current best guess with uniform noise ρ at each iteration and greedily chooses the best point. We applied the algorithm on the same experimental setup with a maximum of 25 iterations and set $\rho = 0.125$. Our algorithm performed 20 real experiments and was able to return 15% better results. Furthermore, the resulting parameters of the foot optimization had 38% lower cost than the results of the arm optimization, which underlines the importance of foot feedback. It is furthermore interesting to note, that the chosen parameter was not at the border of the domain. Additionally, the evaluations of the Random Search algorithm caused a fall of the robot once, and almost lead to a fall at another time, whereas our approach always evaluated safe parameters.

Similar to Fig. 6.4, the behavior during optimization has been visualized in Fig. 6.5. The three rows correspond to the (top to bottom) query point distribution, the GP posterior and the hypothetical posterior without simulations. Like desired, the impact of the simulation has been increased, which can be seen clearly at each iteration. This lead to a large portion of the domain being unexplored, since high simulator costs ruled out large regions. This behavior would not have been possible without simulations, like visualized by the difference of the middle and the bottom row graphs.

Moreover, by evaluating this experiment, we have shown that it is possible to achieve a high impact of the simulation on the target function by adapting hyperparameters. Combining this with the results of Sec. 6.2.1, we have shown that it is possible to achieve both, low and high influence of the simulator. Thus, proper tuning of the hyperparameters can lead to a good compromise of both, and builds the basis for future work.

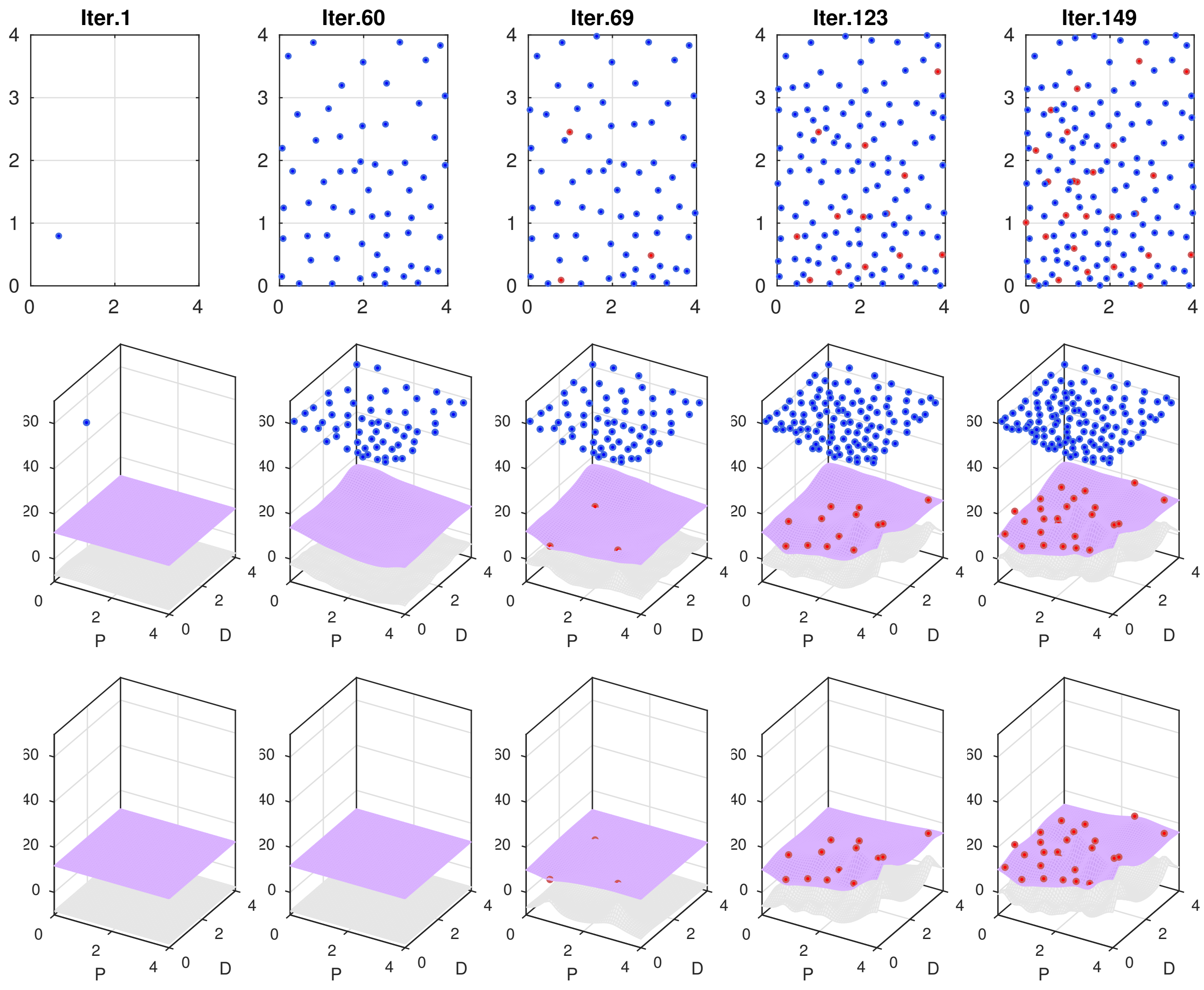

Figure 6.4: The Gaussian Process posterior of the **ArmAngleY** optimization over different iterations. The upper covariance bound has been removed from all plots for better visibility of the mean, but the lower bound is shown as a grey mesh. Blue dots resemble simulation queries, red dots real experiments respectively. The purple mesh displays the mean estimate over the target function. (top) The different queries are shown in the parameter space. (middle) The GP posterior that has been used during the optimization process. (bottom) The GP posterior without simulator-data.

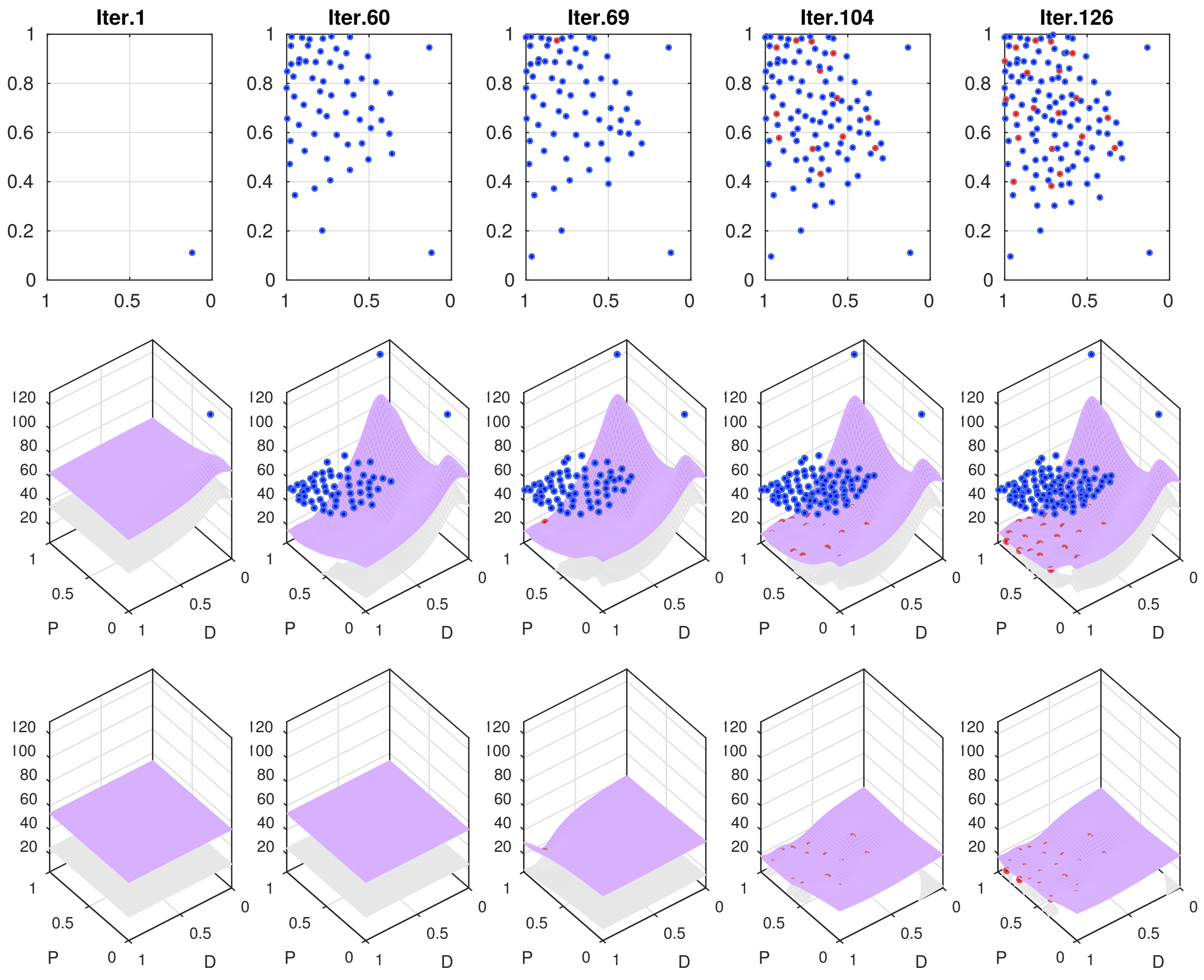

Figure 6.5: The Gaussian Process posterior of the **FootAngleY** optimization over different iterations. The upper covariance bound has been removed from all plots for better visibility of the mean, but the lower bound is shown as a grey mesh. Blue dots resemble simulation queries, red dots real experiments respectively. The purple mesh displays the mean estimate over the target function. Note that all plots have been rotated around the Z-axis for better visibility. (top) The different queries are shown in the parameter space. (middle) The GP posterior that has been used during the optimization process. (bottom) The hypothetical GP posterior without simulator-data.

```
Data: Compact interval $\mathbf{X} \subset \mathbb{R}^d$, search granularity $\rho > 0$, max.
      iterations $n$
Result: Estimated optimum $\hat{\boldsymbol{\theta}} \in \mathbf{X}$
Initialize random $\boldsymbol{\theta}_{best} \in \mathbf{X}$;
$y_{best} = \infty$;
while $i < n$ do
    $y = J(\theta)$;
    if $y < y_{best}$ then
        $\boldsymbol{\theta}_{best} = \boldsymbol{\theta}$;
        $y_{best} = y$;
    end
    $\boldsymbol{\theta} = random(\boldsymbol{\theta}_{best} \pm \rho)$;
    Bound $\boldsymbol{\theta}$ to $\mathbf{X}$;
end
return $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}_{best}$;
```

Algorithm 1: Random search. The algorithm corrupts the current best guess in each iteration and evaluates the resulting point.

6.2.3 4D Optimization

In the second configuration, we optimized 4 parameters, namely the ArmAngleY P- and D-gain together with the FootAngleY P- and D- gain, since these parameters have been observed to be crucial for a robust gait. In the optimization and evaluation only the corrective actions that get optimized are enabled. The setup was stopped by the user after a total of 301 iterations, consisting of 271 simulations and 30 real world experiments. The run was stopped since it was not possible to achieve the initially targeted 50 real world experiments in a time-frame of 10 hours. Nevertheless, the algorithm was able to deduce enough information from the performed simulations to produce reasonable results in the 4D optimization.

The resulting parameters were evaluated using the same setup that was used during optimization (see Section 6.1), performed 15 times. The optimized gait had an average cost of 10.38 and resulted in contrast to the manually tuned gait with a mean cost of 16.28 in an improvement of 35.25%, like shown in Figure 6.6. In order to achieve a better evaluation of the gait stability, we also compared the costs without the penalty function. The resulting performance of the optimized parameters was 53.22% better than the default parameters.

Moreover, we performed a side-by-side comparison of the measured Fused Angle Deviation over time during 5 trials (Fig. 6.7). We also show the average of the absolute deviations together with its integral in Fig. 6.9. Please recall from Section 5.1 that this integral is not the same as the cost function, as the cost function uses the Fused Feedback which is different from the Fused Angle deviations like described in Chapter 3.

In Fig. 6.7, you can see there is no big difference between the optimized and the hand tuned parameters during the first part of the graphs, because the robot was walking on the spot during that time, which typically is very stable. However, as soon as the robot starts walking, the deviations start to diverge and the difference between the parameters becomes apparent. While the optimized

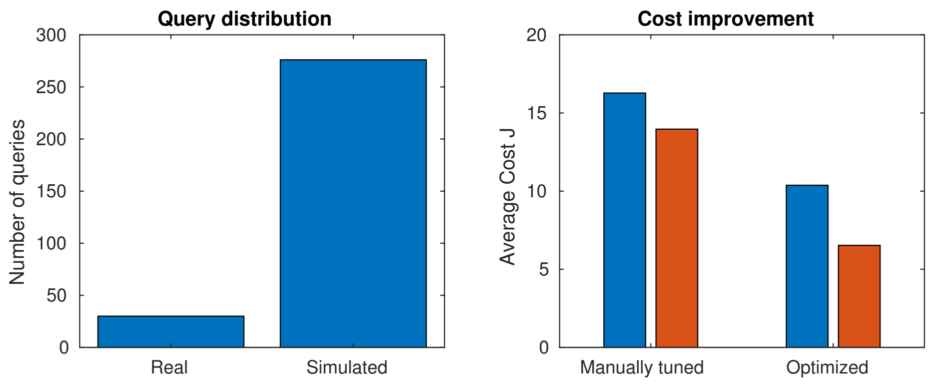

Figure 6.6: (left) 4D Optimization only using 30 real experiments (9.8% of total evaluations) thanks to the information gained through simulations. (right) The average sagittal cost of 15 evaluation is displayed, with (red) and without (blue) the penalty function. The algorithm chose larger parameters than previously, which were performing significantly better.

parameters reproducibly generate deviations of lower amplitude, the manually tuned parameters have partially large deviations and higher variance. This can be explained through the inability of the robot to recover fully from disturbance: Since the robot is not able to fully recover, the disturbances propagate through the trial. Furthermore, when performing test runs with longer duration (25 seconds instead of 15), these deviations occasionally lead to the robot falling with manually tuned parameters, which did not occur with the optimized values.

Furthermore, the mean absolute deviation $\bar{d_\alpha} = mean\,(\|d_\alpha\|)$ and its integral $\bar{D}_\alpha = \int_0^T \bar{d}_\alpha dt$ are displayed in Fig. 6.9. It becomes most apparent that, once the robot has started walking, the amplitude of the deviations are typically lower for the optimized parameters. This effect is visible in the integral of the absolute average, where the two graphs visibly diverge, in which the manually tuned parameters perform worse than the optimized parameters. This observation can also be made in Fig. 6.10, where the robot was performing five longer tests of 25 seconds. It is again possible to see, that the deviations of the default parameters are typically higher. The same observations can be made in the phase plot of the deviations, like depicted in Fig. 6.8. During most of the trials, the hand tuned parameters lead to higher disturbances than the optimized parameters, which are displayed by arches far from the center of the plot.

Additionally, we evaluated the parameters by performing experiments on a field with small obstacles (see Fig. 6.11). During the trial, the robot steps on these obstacles and thus the gait is disturbed. Despite a large deviation, the robot regains stability, which can be also seen in the phase plot of the deviations (see Fig. 6.12). The large arch on the right hand side corresponds to the disturbance at around 10 seconds in the plot over time. This disturbance returns to a stable region near the origin, meaning the robot has stabilized. Furthermore it can be observed, that the deviation is biased. This is because the robot is slightly tilted by the obstacles constantly.

Moreover, the resulting parameters were also qualitatively convincing, when comparing the visual appearance of the gait. More precisely, the optimized gait looked more stable and generally walked with a more upright torso compared to the manually tuned parametrization like shown in Fig. 6.13. A video of these

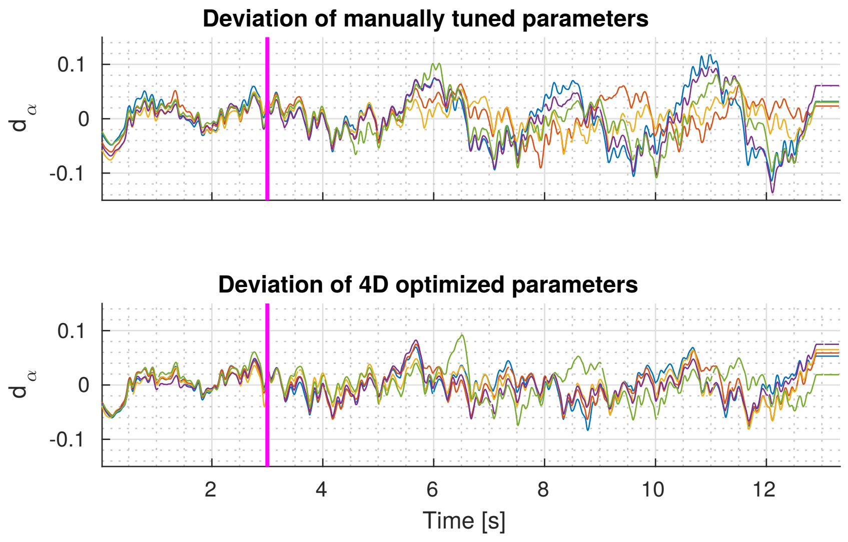

Figure 6.7: A comparison between manually tuned (top) and optimized (bottom) parameters during five real-world experiments. Each curve shows the development of the Fused Angle deviation over time in an experiment. The vertical magenta line indicates the time at which the robot starts walking forward.

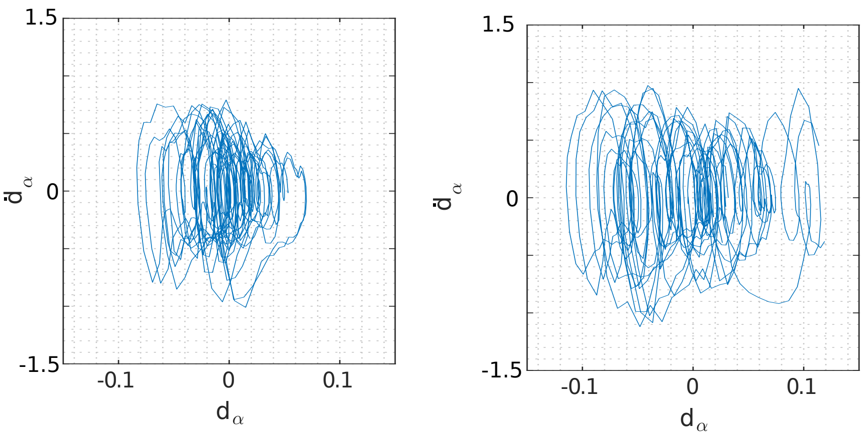

Figure 6.8: The deviation of a single pair of parameter is displayed as a phase plot (comp. Fig. 6.7). The X axis denotes the deviations, whereas at the Y axis its derivative is shown. It is possible to see, that the manually tuned parameters (right) lead to higher disturbances during straight walking compared to the optimized parameters (left).

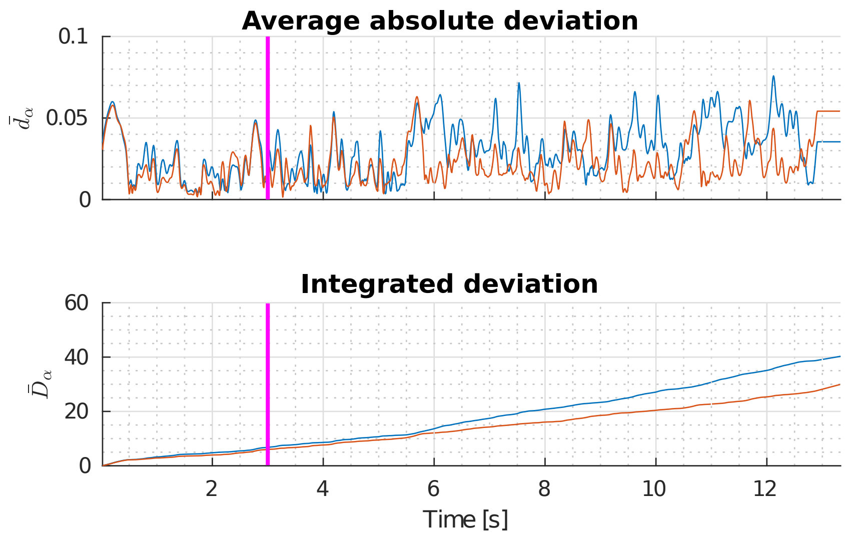

Figure 6.9: The average absolute deviation $\bar{d}_\alpha$ over five experiments of the optimized and default parameters are displayed in the upper figure. The integral $\bar{D}_\alpha$ of these values is shown in the lower figure. These values are for evaluation purpose only and can not be directly compared to the cost, since the latter is constructed of the Fused Feedback (see Chapter 3). The vertical magenta line indicates the time at which the robot starts walking forward. The blue line resembles the hand tuned parameters, the red line the optimized parameters respectively.

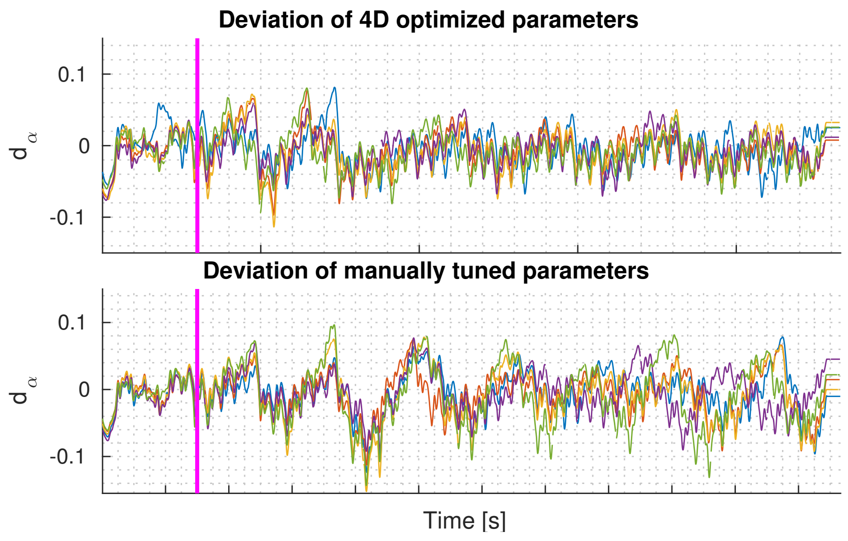

Figure 6.10: A comparison between manually tuned (top) and optimized (bottom) parameters during five longer real-world experiments, where the robot was walking 20 seconds instead of 10. Each curve shows the development of the Fused Angle deviation over the 25 seconds of the experiment. The vertical magenta line indicates the time at which the robot starts walking forward.

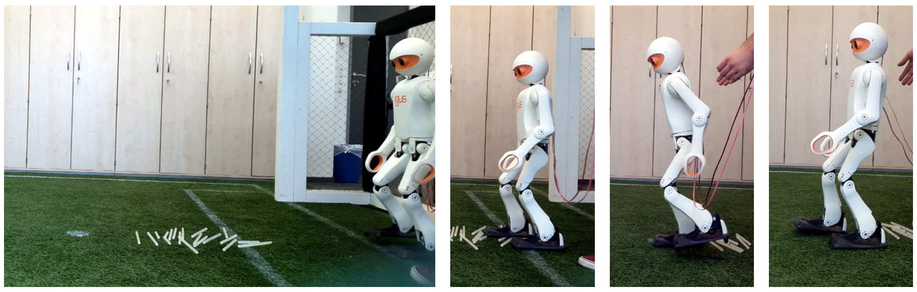
Figure 6.11: The setup of the disturbance test is displayed, where the robot has to walk over small obstacles. It is possible to see, that the disturbance destabilizes the robot temporarily.

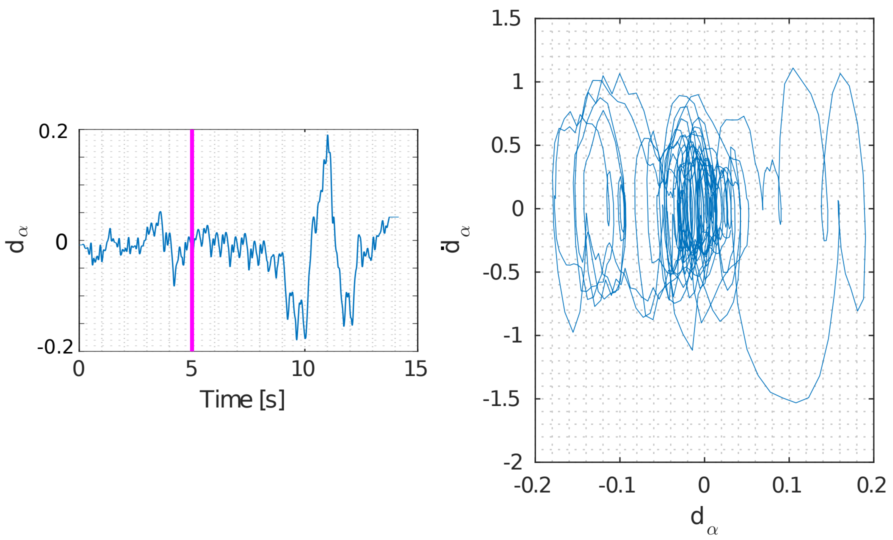

Figure 6.12: (left) The gait deviation is displayed over time. The robot stepped on an obstacle during walking, which is clearly visible in the graph. The magenta line indicates the approximate moment of contact with the obstacles. (right) The phase plot of the corresponding deviations.

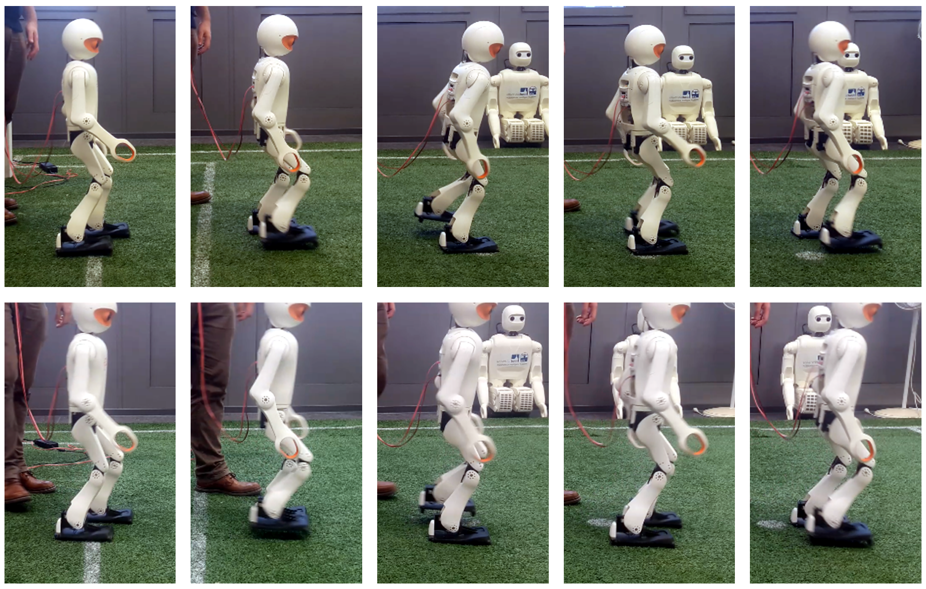

Figure 6.13: The igus® Humanoid Open Platform is performing gait evaluations. The top sequence shows the manually tuned gait, whereas at the bottom the resulting gait of the 4D optimized parameters is displayed. Qualitatively, it is possible to see that the magnitude of the robots tilt is smaller in the top sequence.

results is available online[1].

Another remarkable property of the applied approach is shown by the fact that the robot did not fall a single time during the optimization process. This shows that the algorithm was successfully generalizing the information gathered from the simulation, such that parameters that resulted in a fall in simulation and thus in a higher cost, were ruled out without the need of physical experiments. Furthermore, this generalization also leads to a lower amount of required physical experiments, which underlines the success of this approach.

6.2.4 KLD-threshold

The KLD-threshold criterion was evaluated in a 2D simulation-only experiment. The best parameter estimation after the KLD-threshold was applied (261 iterations) had a resulting mean cost of 111.4604. This performance was compared with the parameter estimation resulting after letting the algorithm run until the Kernel matrix becomes singular, which was after 624 iterations. Since small errors in the prior model become more relevant when the algorithm approximates regions with higher resolution, i.e. after a larger amount of iterations, these errors can lead to bad assumptions. Presumably, this is the reason why the robot was not able to walk with the parameters resulting after 624 iterations. In contrary to this, the robot was able to walk with the parameters at the KLD induced termination. Moreover, the resulting gait was qualitatively convincing and the robot was able to walk stable (see Fig. 6.1).

Undoubtedly, this result underlines the importance of a proper termination condition and shows that our approach is able to deliver a satisfactory criterion.

[1]http://www.ais.uni-bonn.de/videos/RoboCup_Symposium_2018

Chapter 7

Conclusion

In the scope of this bachelor thesis, we implemented a method that trades off robot simulations and real-world experiments to optimize the parameters of the Fused Feedback mechanisms in a sample-efficient manner. Compared to manual tuning, our approach showed a remarkable improvement in gait stability and was able to learn a 4D parametrization without making the robot fall during optimization.

There are still multiple adaptions that can improve the performance of our method. Firstly, it is possible to incorporate the bias of the cost function into the mean function, opposed to the constant mean function that is currently used. This point will have an impact on the sample efficiency, because the use of more prior data might relieve the algorithm, such that it does not have to learn that data. This extension will most likely result in a reduced amount of simulations.

Additionally, if the goal is to learn parameters for robots with the same physical model in the simulator, it is possible to store the Gaussian Process until the first physical experiment is about to be performed. This way it will be possible to reduce the required time for the optimization process for the next robot.

Moreover, we are making efforts to reduce the amount of noise inside the simulator, such that we could remove the averaging inside the cost function. This again would reduce the amount of required time drastically.

A different problem of the method is the low dimensionality that can be currently addressed. We hypothesize to solve this issue by introducing a dimensionality reduction and by optimizing in a lower-dimensional space.

Another improvement could be achieved by learning the hyperparameters of the kernel during optimization.

This approach successfully used information gained through simulations and thus reduced the required amount of real-world experiments drastically. Moreover, using this approach, after 30 real-world experiments we achieved an improvement of more than 50% compared to the prior parameters and thus were able to prove the success of our implementation. Furthermore, we proposed a termination criterion based on the Kullback-Leibler-Divergence, underlined its importance and we demonstrated that the parameters at the point of termination lead to a stable gait.

Bibliography

[1] Riad Akrour et al. “Local Bayesian Optimization of Motor Skills”. In: *Proceedings of the 34th International Conference on Machine Learning.* Proceedings of Machine Learning Research (PMLR). International Convention Centre, Sydney, Australia, 2017, pp. 41–50.

[2] Philipp Allgeuer and Sven Behnke. “Fused Angles: A Representation of Body Orientation for Balance”. In: *Int. Conf. on Intelligent Robots and Systems (IROS).* Hamburg, Germany, 2015.

[3] Philipp Allgeuer and Sven Behnke. “Omnidirectional Bipedal Walking with Direct Fused Angle Feedback Mechanisms”. In: *Proceedings of 16th IEEE-RAS Int. Conference on Humanoid Robots (Humanoids).* Cancún, Mexico, 2016.

[4] Philipp Allgeuer et al. “The igus Humanoid Open Platform: A Child-sized 3D Printed Open-Source Robot for Research”. In: *Künstliche Intelligenz* 30.3 (2016).

[5] R. Calandra et al. “Bayesian Gait Optimization for Bipedal Locomotion”. In: *Proceedings of the 8th International Conference on Learning and Intelligent Optimization.* Lecture Notes in Computer Science. Springer, 2014, pp. 274–290.

[6] T. M. Cover. “Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition”. In: *IEEE Transactions on Electronic Computers* EC-14.3 (1965), pp. 326–334.

[7] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. “Gaussian processes for data-efficient learning in robotics and control”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.2 (2015), pp. 408–423.

[8] Hao Dong, Mingguo Zhao, and Naiyao Zhang. “High-speed and energy-efficient biped locomotion based on Virtual Slope Walking”. In: *Autonomous Robots* 30.2 (2011), pp. 199–216.

[9] Johannes Englsberger et al. “Overview of the torque-controlled humanoid robot TORO”. In: *14th IEEE-RAS International Conference on Humanoid Robots.* 2014, pp. 916–923.

[10] Felix Faber and Sven Behnke. “Stochastic optimization of bipedal walking using gyro feedback and phase resetting”. In: *2007 7th IEEE-RAS International Conference on Humanoid Robots.* 2007, pp. 203–209.

[11] Hafez Farazi et al. “RoboCup 2016 Humanoid TeenSize Winner NimbRo: Robust Visual Perception and Soccer Behaviors”. In: *RoboCup 2016: Robot World Cup XX*. 2017, pp. 478–490.

[12] Elco Heijmink et al. “Learning optimal gait parameters and impedance profiles for legged locomotion”. In: *IEEE-RAS 17th International Conference on Humanoid Robotics*. 2017, pp. 339–346.

[13] P. Hennig and CJ. Schuler. “Entropy Search for Information-Efficient Global Optimization”. In: *Journal of Machine Learning Research* (2012), pp. 1809–1837.

[14] Kazuo Hirai et al. “The development of Honda humanoid robot”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 1998, pp. 1321–1326.

[15] Shuuji Kajita et al. *Introduction to humanoid robotics*. Vol. 101. Springer, 2014. Chap. 3.

[16] Petar Kormushev et al. “Learning to exploit passive compliance for energy-efficient gait generation on a compliant humanoid”. In: *Autonomous Robots* (2018).

[17] Alonso Marco et al. “Automatic LQR tuning based on Gaussian process global optimization”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 270–277.

[18] Alonso Marco et al. “Virtual vs. Real: Trading Off Simulations and Physical Experiments in Reinforcement Learning with Bayesian Optimization”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 1557–1563.

[19] M. Missura and S. Behnke. “Omnidirectional capture steps for bipedal walking”. In: *13th IEEE-RAS International Conference on Humanoid Robots*. 2013, pp. 14–20.

[20] Marcell Missura and Sven Behnke. “Online learning of bipedal walking stabilization”. In: *KI-Künstliche Intelligenz* 29.4 (2015), pp. 401–405.

[21] Marcell Missura and Sven Behnke. “Online learning of foot placement for balanced bipedal walking”. In: *14th IEEE-RAS International Conference on Humanoid Robots*. 2014, pp. 322–328.

[22] Marcell Missura and Sven Behnke. “Self-stable Omnidirectional Walking with Compliant Joints”. In: *8th Workshop on Humanoid Soccer Robots*. IEEE-RAS International Conference on Humanoid Robots, 2013.

[23] Akshara Rai et al. “Bayesian Optimization Using Domain Knowledge on the ATRIAS Biped”. In: *CoRR* abs/1709.06047 (2017).

[24] CE Rasmussen and CKI Williams. *Gaussian Processes for Machine Learning*. en. Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press, 2006, p. 248.

[25] Thomas Röfer. “Evolutionary Gait-Optimization Using a Fitness Function Based on Proprioception”. In: *RoboCup 2004: Robot Soccer World Cup VIII*. Berlin, 2005, pp. 310–322.

[26] Russ Tedrake et al. “A closed-form solution for real-time ZMP gait generation and feedback stabilization”. In: *IEEE-RAS 15th International Conference on Humanoid Robots*. 2015, pp. 936–940.