

---

# Trajectory Kernel for Bayesian Optimization

---

Bachelor-Thesis von Sebastian Rinder aus Sindelfingen  
Februar 2018



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



## Trajectory Kernel for Bayesian Optimization

Vorgelegte Bachelor-Thesis von Sebastian Rinder aus Sindelfingen

1. Gutachten: Prof. Dr. N. N.
2. Gutachten: Prof. Dr. N. N.
3. Gutachten: Prof. Dr. N. N.

Tag der Einreichung:

Please cite this document with:

URN: [urn:nbn:de:tuda-tuprints-38321](https://nbn-resolving.org/urn:nbn:de:tuda-tuprints-38321)

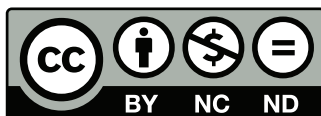
URL: <http://tuprints.ulb.tu-darmstadt.de/id/eprint/3832>

Dieses Dokument wird bereitgestellt von tuprints,

E-Publishing-Service der TU Darmstadt

<http://tuprints.ulb.tu-darmstadt.de>

[tuprints@ulb.tu-darmstadt.de](mailto:tuprints@ulb.tu-darmstadt.de)



This publication is licensed under the following Creative Commons License:

Attribution – NonCommercial – NoDerivatives 4.0 International

<http://creativecommons.org/licenses/by-nc-nd/4.0/>

---

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

Hiermit versichere ich, Sebastian Rinder, 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.

---

### **English translation for information purposes only:**

#### **Thesis Statement pursuant to § 23 paragraph 7 of APB TU Darmstadt**

I herewith formally declare that I, Sebastian Rinder, 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.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

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

Datum / Date:

Unterschrift / Signature:

---

---

---

## Abstract

Reinforcement learning relies on policy gradient but the gradient is known only in expectation and most of the time stochastic policies. This leaves some room for zero order methods and BO can combine solving the problem and the exploration strategy from deterministic policies. We investigate in this paper how to integrate efficient exploration strategies stemming from Bayesian optimization for solving high dimensional reinforcement learning problems. We propose a novel optimization algorithm that is able to scale Bayesian optimization to such high dimensional tasks by restricting the search to the local vicinity of a search distribution and by proposing kernels capturing similarity in behavior rather than parameter. We show in the experiments that our approach can be very useful for applications such as robotics.

## Zusammenfassung

Das Ziel im bestärkten Lernen ist das Finden einer Strategie, welche die erhaltene Belohnung eines Agenten maximiert. Da der Suchraum für mögliche Strategien sehr groß sein kann, verwenden wir Bayesian optimization, um die Anzahl der Evaluierungen durch den Agenten zu minimieren. Das hat den Vorteil, dass zeit- und kostenaufwändige Abläufe, wie beispielsweise das Bewegen eines Roboterarms, reduziert werden. Die Effektivität der Suche wird maßgeblich von der Wahl des Kernels beeinflusst. Standardkernel in der Bayesian optimization vergleichen die Parameter von Strategien um eine Vorhersage über bisher nicht evaluierte Strategien zu treffen.

Der Trajectorykernel vergleicht statt der Parameter, die aus den jeweiligen Strategien resultierenden Verhaltensweisen. Dadurch werden unterschiedliche Strategien mit ähnlichem Resultat von der Suche weniger priorisiert.

Wir zeigen die Überlegenheit des verhaltensbasierten Kernels gegenüber dem parameterbasierten anhand von Roboters-terierungssimulationen.

As a consequence the trajectory kernel makes the search process more efficient, but also has the disadvantage that the effort for kernel calculations is greatly increased.

The trajectory kernel relates policies derives a behaviour measure from the state and action parameter sets, to accordingly. This relation has the advantage that different policies with similar results are recognized as redundant and are therefore less prioritized by the search. As a consequence the trajectory kernel makes the search process more efficient, but also has the disadvantage that the effort for kernel calculations is greatly increased.

When it comes to higher dimensional problems Bayesian optimization tends to over explore because of its focus on global optimization. This can lead to not finding the optimum. We therefore also use Bayesian optimization with a locally restricted search area [1]. This area is adjusted throughout the search process to cover the most promising parts of the entire search space. As a result, we only have to optimize locally. This method is more robust against high-dimensional problems and also computationally less demanding than global optimization.

So we will compare the trajectory kernel to standard kernels in the Bayesian optimization. We also do test runs with global Bayesian optimization and Bayesian optimization in a local context. The reinforcement learning environment will be given by the classic control tasks cart pole, mountain car and acrobot [2].

---

## Acknowledgments

---

# Contents

<b>1. Introduction</b>	<b>2</b>
<b>2. Foundations</b>	<b>3</b>
2.1. Bayesian optimization . . . . .	3
2.1.1. Gaussian Process Regression . . . . .	3
2.1.2. Hyper parameter optimization . . . . .	5
2.1.3. Acquisition function . . . . .	5
2.2. Reinforcement learning . . . . .	6
2.2.1. Markov decision process . . . . .	7
2.2.2. Trajectory kernel . . . . .	7
<b>3. Contributions</b>	<b>9</b>
3.1. Local Bayesian optimization . . . . .	9
3.1.1. Trajectory kernel . . . . .	9
3.2. Numerical stability and efficiency . . . . .	10
3.2.1. Gaussian Process Regression . . . . .	10
3.2.2. Action selection . . . . .	11
3.2.3. Hyper parameter optimization . . . . .	11
<b>4. Experiments</b>	<b>13</b>
4.1. Implementation . . . . .	13
4.1.1. Efficiency . . . . .	13
4.1.2. Global optimization . . . . .	13
4.1.3. Optimization starting point . . . . .	13
4.1.4. Problem environment . . . . .	13
4.1.5. Action selection . . . . .	13
4.2. Start conditions . . . . .	14
4.3. Environments . . . . .	14
4.3.1. Cart Pole . . . . .	14
4.3.2. Acrobot . . . . .	15
4.3.3. Mountain car . . . . .	15
<b>5. Results</b>	<b>16</b>
<b>6. Discussion</b>	<b>17</b>
<b>7. Outlook</b>	<b>18</b>
<b>Bibliography</b>	<b>19</b>
<b>A. Some Appendix</b>	<b>21</b>

# Figures and Tables

---

## List of Figures

---

2.1. Visualization of the change in the Gaussian process model and the resulting acquisition function. With every Bayesian optimization iteration step a new observation point at the maximum of the acquisition function is added to the training data. This new observation and its evaluation update the Gaussian process model. Thus the model becomes increasingly accurate in the vicinity of each observation. [3] . . . 4

2.2. Visualization of Gaussian process posterior values corresponding to different acquisition functions. In the upper frame the posterior mean  $\mu$  and double standard deviation  $2\sigma$  is marked as the blue line with the shaded area. The black dashed line shows the true objective and the red crosses mark already evaluated points. The lower frame compares the plots of the resulting acquisition functions probability of improvement, expected improvement, upper confidence bounds, Thompson sampling and predictive entropy search. The peaks of probability of improvement next to values with high mean illustrates its exploitive behaviour compared to expected improvement, which takes the variance more into account. Thompson sampling is the one with the most explorative peak due to its random nature. [4] . . . . . 6

4.1. Visualization of the Cart Pole rendered by the Open AI Gym . . . . . 14

4.2. Visualization of the Acrobot rendered by the Open AI Gym . . . . . 15

4.3. Visualization of the mountain car rendered by the Open AI Gym . . . . . 15

---

## List of Tables

---

4.1. Environment parameters of the experiments . . . . . 14

---

## Symbols and Notation

Matrices and constants are noted as capital letters, and vectors and scalars as lower case letters. Vectors are assumed as column vectors. The first dimension of a matrix indicates its row number, the second dimension its column number.

<u>Symbol</u>	<u>Meaning</u>
$S$	Matrix of states
$A$	Matrix of actions
$p$	state transitioning probabilitiy
$p_0$	probabilitiy of starting in state $s_0$
$r$	immediate reward
$\bar{r}$	cumulative reward, sum of all rewards from one trajectory
$M$	number of initial sample points before starting the Bayesian optimization
$N$	number of total Bayesian optimization steps
$n$	number of training points present
$n_*$	number of sample test points
$d$	number of dimensions in the problem
$x$	training point vector of length $d$
$x_*$	test point vector of length $d$
$X$	$n \times d$ matrix of $n$ points $x^\top$
$X_*$	$n_* \times d$ matrix of $n_*$ test points $x_*^\top$
$y$	vector of $n$ evaluated objective function values
$K(X, X) = K$	$n \times n$ covariance matrix
$K(X, X_*) = K_*$	$n \times n_*$ covariance between training and test points
$K(X_*, X) = K_*^\top$	same as $K(X, X_*)^\top$
$K(X_*, X_*) = K_{**}$	$n_* \times n_*$ covariance matrix
$\circ$	Hadamard product, element-wise product
$\text{diag}(V)$	returns the diagonal elements of the square matrix $V$ as a vector
$\mathcal{O}(n)$	Big O notation - linear algorithm performance
$\mathcal{O}(n^2)$	Big O notation - quadratic algorithm performance



---

# 1 Introduction

The goal of machine learning science is to create well-functioning systems without programming them explicitly. Especially for tasks where manual design results in poor performance. As in image or speech recognition, effective web search, or realization of self-driving cars. Considering the huge amount of data to be processed, machine learning algorithms easily surpass deterministic design. The advantage stems from learning from examples and then generalizing them. This means that patterns in the learning data can be recognized and transferred to future situations without human supervision.

In reinforcement learning, a sub-area of machine learning, we want an agent to teach himself how to improve his behaviour. We want a specific result from that agent, therefore we discourage results we do not seek and reinforce behaviour which leads to our goal. As an agent let us assume a robot arm with a table tennis racket and a ball. If our goal was to balance the ball for as long as possible, we would penalize the agent for letting the ball fall, and reward the agent for each successful time step of balancing. At the end of an evaluation episode we sum up all received rewards to get the cumulative reward as a measure of performance.

The agent's behaviour can be defined by a set of policy parameters. These parameters tell the agent which action to execute depending on the state. A state of the table tennis agent for example would contain the angle of the racket, and the position of the ball. A well tuned policy could tell the arm when to rotate, and at which magnitude, to keep the ball from falling. Finding such a well performing policy is the goal of our reinforcement learning task.

One approach of maximizing the performance of an agent could try random policies until finding a sufficient one. Unfortunately this would take a very long time since it is highly unlikely to find fitting policy parameters on accident. Especially in more complex environments with higher dimensional policies. Besides the huge number of evaluations necessary, we would have to deal with wear and tear, and the limited movement speed of our robotic equipment.

So we require a more efficient search process for finding properly functioning policies. Optimally a search process, which learns from collected data, and extrapolates future behaviours. In recent years the Bayesian optimization approach has been shown to be very efficient in this field [3, 4, 5]. It tries to find an optimum of a black box function in as few steps as possible. In our reinforcement learning task the black box function would be the cumulative reward of an episode generated by a specific policy. And the optimum would be the policy with the highest cumulative reward achievable.

The prediction of future behaviour is based on measuring the distance between already evaluated policies with known results. The outcome of a new policy is then estimated by applying the learned distance metric. Out of many new policies Bayesian optimization selects one with the most promising enhancement. A good enhancement measure will include policies of uncharted areas in the search space and policies with high expected cumulative rewards as well. This balancing act is known as the exploration exploitation trade-off. Only exploiting policies with a high expectation can lead to a local optimum and thus to a result below the best possible one. Whereas excessive exploration will cover the entire search space, but maybe will not find an optimum at all.

When using standard kernels, the distance between policies is usually measured by calculating the Euclidean distance between corresponding policy parameters. A more meaningful measure would be to compare the resulting behaviors stemming from certain policies rather than comparing the parameters of the policies. We incorporate such a behaviour measure using the trajectory data generated by our agent during an evaluation process. This data, consisting of state and action values, is used by the trajectory kernel to relate the respective policies.

---

## 2 Foundations

---

### 2.1 Bayesian optimization

---

To find the point of the maximum of an expensively to evaluate black box objective function we use Bayesian optimization. It makes the search process more efficient by incorporating a Gaussian process model to anticipate the objective function's behaviour. This model, containing estimates of the function's returns depending on points, is used by a so called acquisition function to guide the exploration for promising new points. Each selected point is evaluated by the objective function and the results are included in the Gaussian process model. Optimally these steps are repeated until found function values converge at the maximum possible function value. In reality (Algorithm 1) we iterate over  $N$  steps and analyse the results. The change of the Gaussian process model affecting the acquisition function during Bayesian optimization iterations is illustrated in Figure 2.1.

```
input :  $X$ :  $M$  uniformly random samples from the search space  
         $N$ : number of Bayesian optimization iterations  
        set hyper parameters to 1  
output:  $x_{M+N}$   
 $y$  = evaluations of the objective at the training points  $X$   
for  $n = M$  to  $M + N$  do  
    optimize hyper parameters (optional)  
    prepare a Gaussian process model depending on  $X$ ,  $y$  and hyper parameters  
     $x_{n+1}$  = point at the optimum of the acquisition function  
     $y_{n+1}$  = evaluation of the objective at the point  $x_{n+1}$   
     $X = \{X, x_{n+1}\}$   
     $y = \{y, y_{n+1}\}$ 
```

**Algorithm 1:** Global Bayesian optimization

The optimization of hyper parameters is optional and depends on the Gaussian process kernel as mentioned in chapter 6. Also the noise level parameter  $\sigma_n$  is not considered as hyper parameter in this context. It is tuned separately 5.

---

#### 2.1.1 Gaussian Process Regression

---

The Gaussian process finds a prior distribution over the possible functions that are consistent with the training data. From the regression we then get a posterior mean and variance, which describe our model of the objective function. The mean represents a prediction of the true objective at a given point and the variance represents the uncertainty at that point. The more training points our model incorporates the smaller the variance, and the preciser the predictions, in the proximity around training points.

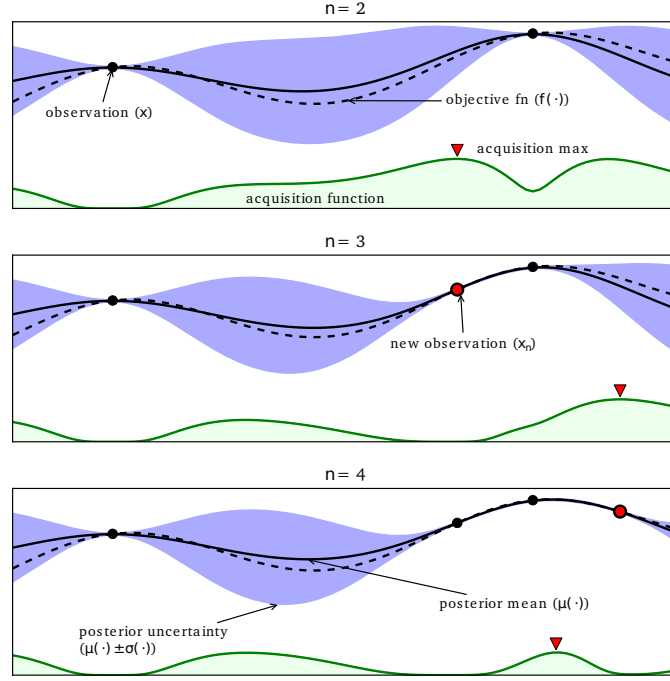
In real world applications we always have some noise in the objective observations. Therefore a Gaussian distributed error term,

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2),$$

with zero mean and  $\sigma_n^2$  variance is added to the function value. Ergo the observed target

$$y = f(x) + \epsilon$$

regards this noise. The knowledge our training data provides is represented by the covariance matrix  $K(X, X)$ . With the matrix of test points  $X_*$  we get the joint distribution of the target values and the estimated function values at the test locations:



**Figure 2.1.:** Visualization of the change in the Gaussian process model and the resulting acquisition function. With every Bayesian optimization iteration step a new observation point at the maximum of the acquisition function is added to the training data. This new observation and its evaluation update the Gaussian process model. Thus the model becomes increasingly accurate in the vicinity of each observation. [3]

$$\begin{bmatrix} y \\ f_* \end{bmatrix} \sim \mathcal{N} \left( 0, \begin{bmatrix} K(X, X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix} \right).$$

For further simplification we define  $K = K(X, X)$ ,  $K_* = K(X, X_*)$ ,  $K_*^\top = K(X, X_*)^\top = K(X_*, X)$ , and  $K_{**} = K(X_*, X_*)$ . Now we can calculate the posterior mean and variance at given test points  $X_*$ :

$$K_n = K + \sigma_n^2 I \quad (2.1)$$

$$\mu = K_* K_n^{-1} y \quad (2.2)$$

$$V(x_*, x_*) = K_{**} - K_* K_n^{-1} K_*^\top \quad (2.3)$$

$$v(x_*) = \text{diag}(V) \quad (2.4)$$

$$\sigma(x_*) = \sqrt{v(x_*)}. \quad (2.5)$$

The resulting vectors  $\mu$  and  $v$  contain the means and variances for all corresponding test points. Also we get the whole covariance matrix  $V$ .

### 2.1.1.1 Standard kernel

The similarity between points is measured by the covariance function. The better this covariance function is suited for our objective function the more precise is the resulting model. We use two standard kernels to compare them to the trajectory kernel (section 2.2.2). In those standard kernels the distance metric for two points is given by the Euclidean distance:

$$D(x_i, x_j) = (x_i - x_j)^\top (x_i - x_j).$$

Which we then use in the squared exponential kernel,

$$K(x_i, x_j, \sigma) = \sigma_f^2 \exp\left(-\frac{D(x_i, x_j)}{2\sigma_l^2}\right),$$

and the Matern 5/2 kernel,

$$K(x_i, x_j, \sigma) = \sigma_f^2 \left(1 + \frac{\sqrt{5D}}{\sigma_l} + \frac{5D}{3\sigma_l^2}\right) \exp\left(-\frac{\sqrt{5D}}{\sigma_l}\right),$$

where  $\sigma_f$  denote the signal standard deviation and  $\sigma_l$  the characteristic length scale. These so called hyper parameters can be tuned by hand or set with hyper parameter optimization.

---

### 2.1.2 Hyper parameter optimization

---

–EXPLAIN MARGINAL likelihood.

Selecting proper hyper parameters for the Gaussian process regression can reduce the number of objective function evaluations necessary. Also it helps avoiding numerical problems. To find an optimum for the signal deviation hyperparameter  $\sigma_f$  and the length scale hyperparameter  $\sigma_l$  we maximize the log marginal likelihood function

$$\log p(y|X, \sigma_f, \sigma_l) = -\frac{1}{2}y^\top K_n^{-1}y - \frac{1}{2}\log |K_n| - \frac{n}{2}\log 2\pi, \quad (2.6)$$

of the Gaussian process. The number of observations is  $n$ ,  $X$  is the  $d \times n$  dataset of inputs and  $K_n$  is the covariance matrix for the noisy target  $y$ .

---

### 2.1.3 Acquisition function

---

The basis of the Bayesian optimization consists of selecting the next evaluation point in our search space. This point is at the optimum of the acquisition function, which depends on the current Gaussian process model. We choose expected improvement during the global Bayesian optimization and in the local optimization (chapter 3) we use Thompson sampling. Both acquisition functions take the mean und the variance generated by the Gaussian process model into account to guide the exploration process. The difficulty lies in avoiding excessive exploration or exploitation. Exploration seeks points with a high variance and exploitation selects points with a high mean instead. The latter one would result in a local optimum whereas too much exploration may not improve at all. An exemplary comparison of acquisition function is shown in Figure 2.2.

---

#### 2.1.3.1 Expected improvement

---

To get an expected improvement function value at a test point  $x_*$ , we need the mean value  $\mu(x_*)$ , and the standard deviation value  $\sigma(x_*) = \sqrt{v(x_*)}$  from the Gaussian process model. Also we need the maximum of all observations  $y_{max}$  and a trade-off parameter  $\tau$ . For the cumulative distribution function we write  $\Phi(\cdot)$  and for the probability density function we write  $\phi(\cdot)$ . They are both Gaussian with zero mean and unit variance. We adopt the expected improvement function

$$EI(x_*) = \begin{cases} (\mu(x_*) - y_{max} - \tau)\Phi(z(x_*)) + \sigma(x_*)\phi(z(x_*)) & \text{if } \sigma(x_*) > 0 \\ 0 & \text{if } \sigma(x_*) = 0 \end{cases}$$

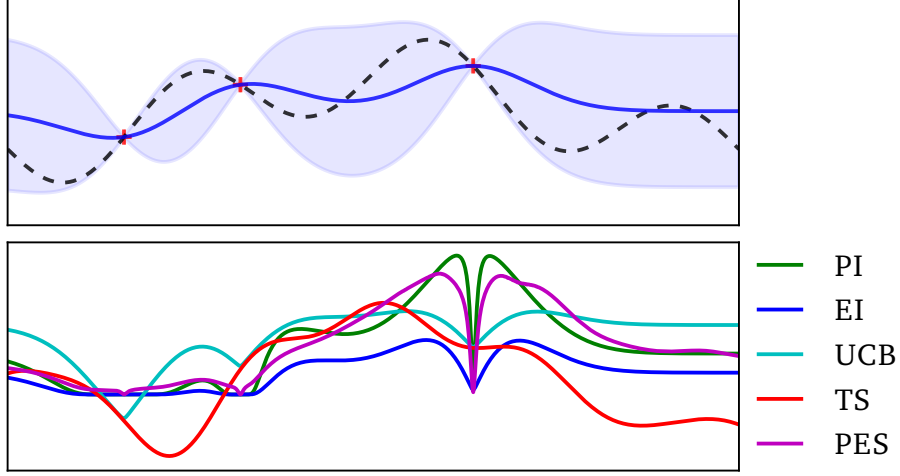
where

$$z(x_*) = \begin{cases} \frac{(\mu(x_*) - y_{max} - \tau)}{\sigma(x_*)} & \text{if } \sigma(x_*) > 0 \\ 0 & \text{if } \sigma(x_*) = 0 \end{cases}$$

as suggested in [3]. The trade-off parameter  $\tau$  is set to 0.01 accordingly. To get the next evaluation point,

$$x_{n+1} = \arg \max_{x_*} EI(x_*),$$

we optimize the expected improvement function over the whole search space.



**Figure 2.2.:** Visualization of Gaussian process posterior values corresponding to different acquisition functions. In the upper frame the posterior mean  $\mu$  and double standard deviation  $2\sigma$  is marked as the blue line with the shaded area. The black dashed line shows the true objective and the red crosses mark already evaluated points. The lower frame compares the plots of the resulting acquisition functions probability of improvement, expected improvement, upper confidence bounds, Thompson sampling and predictive entropy search. The peaks of probability of improvement next to values with high mean illustrates its exploitive behaviour compared to expected improvement, which takes the variance more into account. Thompson sampling is the one with the most explorative peak due to its random nature. [4]

### 2.1.3.2 Thompson sampling

For Thompson sampling acquisition values we sample one function from the Gaussian process posterior,

$$TS \sim GP(0, K(X, X_*)),$$

where  $X$  is the dataset of already evaluated points, and  $X_*$  is a randomly Gaussian distributed set of points with mean and variance given by the local optimizer. These mean and variance represent our current search space. To draw function values we need the mean vector  $\mu$  and the full covariance matrix  $V$  from the Gaussian process model. First we take the lower Cholesky decomposite of  $V$  such that  $L_V L_V^T = V$ . Then we generate a vector  $g$ , which consists of independent Gaussian distributed values with zero mean and unit variance. Finally we get a vector  $TS$  of sampled values:

$$TS(x_*) = \mu + L_V g.$$

We take the one with the highest value such that,

$$x_{n+1} = \arg \max_{x_*} TS(x_*),$$

to get the next evaluation point.

## 2.2 Reinforcement learning

In our reinforcement learning task we want to maximize the cumulative reward stemming from a policy with as few evaluations as feasible. We achieve this maximization by applying Bayesian optimization, treating the cumulative reward as the objective function and policies as points. The objective function evaluation consists of the simulation of our robotic agent.

---

### 2.2.1 Markov decision process

---

We use the Markov decision process model to describe the agent's decisions making [2]. This model consists of a tuple  $(S, A, P, R, \gamma)$ , holding all states  $s \in S$ , all actions  $a \in A$ , all state transitioning probabilities  $p \in P$ , and all corresponding rewards  $r \in R$ . The discount factor  $\gamma$  is set to 1 because we have a finite horizon Markov decision process where the timing of the reward is irrelevant. Since the transitioning probabilities and the rewards are unknown, they are accessed through simulation runs of our environment. Assume an agent which executes a policy  $x$  for  $T$  time steps, producing a trajectory  $\xi$ , and therefore receiving the final reward

$$\bar{r}(\xi) = \sum_{t=1}^T \gamma^t r(s_{t-1}, a_{t-1}, s_t),$$

which will be the sum of all immediate rewards given by a rewarding function  $r(s_{t-1}, a_{t-1}, s_t)$ . The explicit probabilities corresponding to actions and state transitionings are also obtained (see 4.1.5) throughout the simulation run.

---

### 2.2.2 Trajectory kernel

---

Standard kernels like the squared exponential kernel, relate policies by measuring the difference between policy parameter values. Therefore policies with similar behavior but different parameters are not compared adequately. The behavior based trajectory kernel fixes this, by relating policies to their resulting behavior. This makes our policy search more efficient, since we avoid redundant search of different policies with similar behaviour.

---

#### 2.2.2.1 Behaviour based measurement

---

For relating policies to their resultThe optimization of hyper parameters is optional and depends on the Gauging behavior we use the Markov decision process transitioning probabilities. We formulate the conditional probability of observing trajectory  $\xi$  given policy  $x$  as proposed in [6]:

$$p(\xi|x) = p_0(s_0) \prod_{t=1}^T p(s_t|s_{t-1}, a_{t-1}) p_\pi(a_{t-1}|s_{t-1}, x).$$

Trajectory  $\xi = (s_0, a_0, \dots, s_{T-1}, a_{T-1}, s_T)$  contains the sequence of state, action tuples and policy  $x$  a set of  $d$  parameters.  $p_0(s_0)$  is the probability of starting in the initial state  $s_0$ .  $p(s_t|s_{t-1}, a_{t-1})$  is the probability of transitioning from state  $s_{t-1}$  to  $s_t$  when action  $a_{t-1}$  is executed. The stochastic mapping  $p_\pi(a_{t-1}|s_{t-1}, x)$  is the probability for selecting the action  $a_{t-1}$  when in state  $s_{t-1}$  and executing the parametric policy  $x$ .

---

#### 2.2.2.2 Distance metric

---

To examine the difference between two policies  $x_i$  and  $x_j$  the discrete Kullback Leibler divergence

$$D_{\text{KL}}(P(\xi|x_i)||P(\xi|x_j)) = \sum_i P(\xi|x_i) \log \frac{P(\xi|x_i)}{P(\xi|x_j)}.$$

is applied to the policy-trajectory mapping probabilities  $P(\xi|x_i)$  and  $P(\xi|x_j)$ . It measures how the two distributions diverge from another.

In general  $D_{\text{KL}}(P(\xi|x_i)||P(\xi|x_j))$  is not equal to  $D_{\text{KL}}(P(\xi|x_j)||P(\xi|x_i))$ . But we need a symmetric distance measure. So we sum up the two divergences

$$D(x_i, x_j) = D_{\text{KL}}(P(\xi|x_i)||P(\xi|x_j)) + D_{\text{KL}}(P(\xi|x_j)||P(\xi|x_i)),$$

to achieve that  $D(x_i, x_j) = D(x_j, x_i)$ . An additional requirement for the kernel is the resulting matrix to be positive semi-definite and scalable[6]. Therefore we exponentiate the negative of our distance matrix  $D$ . We also apply the hyper parameters  $\sigma_f^2$  to compensate for the signal variance and  $\sigma_l^2$  to adjust for signal length scale. This gives us the covariance function

$$K(x_i, x_j, \sigma_f, \sigma_l) = \sigma_f^2 \exp\left(-\frac{D(x_i, x_j)}{2\sigma_l^2}\right). \quad (2.7)$$

The distance metric is adapted from [6]. We only added the signal variance hyper parameter.

---

### 2.2.2.3 Estimation of Trajectory Kernel Values

---

We estimate the divergence values, because the computational effort will be greatly reduced[6]. We use a Monte-Carlo estimate for the approximation

$$\hat{D}(x_i, x_j) = \sum_{\xi \in \xi_i} \log \left( \frac{P(\xi|x_i)}{P(\xi|x_j)} \right) + \sum_{\xi \in \xi_j} \log \left( \frac{P(\xi|x_j)}{P(\xi|x_i)} \right) \quad (2.8)$$

of the divergences between policies with already sampled trajectories. Here  $\xi_i$  is the set of trajectories generated by policy  $x_i$ . For our gaussian process regression we also need a distance measure between a policy with known trajectories and new policies with unknown trajectories. Since there is no closed form solution to this we use the importance sampled divergence

$$\hat{D}(x_{new}, x_j) = \sum_{\xi \in \xi_j} \left[ \frac{P(\xi|x_{new})}{P(\xi|x_j)} \log \left( \frac{P(\xi|x_{new})}{P(\xi|x_j)} \right) + \log \left( \frac{P(\xi|x_j)}{P(\xi|x_{new})} \right) \right]$$

to estimate the divergence between the new policy  $x_{new}$  and the policy  $x_j$  with already sampled trajectories  $\xi_j$ . Since we only have a ratio of transitioning probabilities present in our trajectory kernel we can reduce the logarithmic term to:

$$\begin{aligned} \log \left( \frac{P(\xi|x_i)}{P(\xi|x_j)} \right) &= \log \left( \frac{P_0(s_0) \prod_{t=1}^T P_s(s_t|s_{t-1}, a_{t-1}) P_\pi(a_{t-1}|s_{t-1}, x_i)}{P_0(s_0) \prod_{t=1}^T P_s(s_t|s_{t-1}, a_{t-1}) P_\pi(a_{t-1}|s_{t-1}, x_j)} \right) \\ &= \log \left( \prod_{t=1}^T \frac{P_\pi(a_{t-1}|s_{t-1}, x_i)}{P_\pi(a_{t-1}|s_{t-1}, x_j)} \right) \\ &= \sum_{t=1}^T \log \left( \frac{P_\pi(a_{t-1}|s_{t-1}, x_i)}{P_\pi(a_{t-1}|s_{t-1}, x_j)} \right). \end{aligned}$$

Summing up the logarithms in the end is also numerically more stable than taking the logarithm of the products.

## 3 Contributions

- including trajectory kernel to local BO –TS trajectory kernel

### 3.1 Local Bayesian optimization

- starting search dist - formula for search dist update

Modelling the objective function for a higher dimensional search space is challenging. Also global Bayesian optimization tends to over-explore. To perform a more robust optimization we use local Bayesian optimization as stated in [1]. It restricts the search space of the acquisition function to a local area which is moved, resized, and rotated between iterations. This local area is defined by a Gaussian distribution in which the mean and variance represent the center and the exploration reach respectively. To update that mean and variance properly we minimize the Kullback-Leibler divergence between the incumbent search distribution  $\pi_n$  and the probability  $p_n^* = p(\mathbf{x} = \mathbf{x}^* | \mathcal{D}_n)$  of  $\mathbf{x}^*$  being optimal. This results in a search area which neglects poorly performing regions.

To prevent the mean from moving too fast from the initial point and to avoid the variance becoming too small quickly we constrain the minimization with the hyper parameters  $\alpha$  and  $\beta$ . Therefore our optimization problem is given by

$$\begin{aligned} \arg \min_{\pi} \quad & \text{KL}(\pi || p_n^*), \\ \text{subject to} \quad & \text{KL}(\pi || \pi_n) \leq \alpha, \end{aligned} \quad (3.1)$$

$$\mathcal{H}(\pi_n) - \mathcal{H}(\pi) \leq \beta, \quad (3.2)$$

where  $\text{KL}(p||q) = \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x}$  is the KL divergence between  $p$  and  $q$  and  $\mathcal{H}(p) = - \int p(\mathbf{x}) \log(p(\mathbf{x})) d\mathbf{x}$  is the entropy of  $p$ .

**input** :  $X$ :  $N$  already evaluated points

$X_*$ :  $M$  random samples from the search distribution

**output**:  $x_{n+1}$ : next evaluation point

$m_d$  = mahalanobis distance from test points to incumbent search distribution

discard samples, which are outside 80% of the cumulative inverse-chi-squared distribution

get mean and covariance matrix from the Gaussian process

get values from Thompson sampling

$x_{n+1}$  = sample at the highest Thompson sampled value

**Algorithm 2:** Thompson Sampling acquisition for Local Bayesian optimization

#### 3.1.1 Trajectory kernel

If using Thompson sampling as the acquisition function, the whole covariance matrix  $V$  of test points is needed. To compute this covariance matrix we need a distance measure between unknown trajectories. The kernel proposed by [6] does not support a measurement between unevaluated policies. We therefore implement our own distance metric based on states, which are already present in the training data. First we filter a random subset from the training data and save it to  $S_{sub}$ . For the continuous action space we then calculate each mean

$$\mu_i = f_s(S_{sub})^\top x_i \quad \text{and} \quad \mu_j = f_s(S_{sub})^\top x_j$$

for the given policies  $x_i$  and  $x_j$ . The difference between the two policies is measured by the squared Euclidean distance between the means:

$$D(x_i, x_j) = (\mu_i - \mu_j)^\top (\mu_i - \mu_j).$$

For a discrete action space we sample the actions  $a$  from  $P(a|S_{sub}, x)$  for the given policies  $x_i$  and  $x_j$  from (4.2). The sampled actions then select the associated probabilities. We compare the discrete probability distributions for each policy by applying the symmetric Kullback-Leibler divergence:

$$D(x_i, x_j) = \sum_i P(a|S_{sub}, x_i) \log \frac{P(a|S_{sub}, x_i)}{P(a|S_{sub}, x_j)} + \sum_j P(a|S_{sub}, x_j) \log \frac{P(a|S_{sub}, x_j)}{P(a|S_{sub}, x_i)}.$$



---

### 3.2 Numerical stability and efficiency

---

Due to numerical instabilities, errors may occur during calculations. These errors are caused by the discrete implementation of continuous mathematical formulas. Sometimes numerical instabilities lead to negative values for covariances. To avoid getting complex numbers we assume negative values as zero before applying the square root on covariances when computing the standard deviation from the Gaussian process model (2.5). Negative values can also occur for the distance between two trajectories, and are thresholded to zero too.

When computing the symmetrical distance matrix  $\hat{D}(X, X)$  (2.8) only the upper triangle matrix  $\hat{D}_u$  is calculated. That halves the computational effort. We obtain  $\hat{D} = \hat{D}_u + \hat{D}_u^\top$ , since the diagonal elements of  $\hat{D}$  are zero and  $\hat{D}(x_i, x_j) = \hat{D}(x_j, x_i)$

To gain robustness we use the lower Cholesky decomposition instead doing matrix inverse calculation. Therefore the matrix to decompose must be positive definite. We achieve this by doubling the noise variance added to the diagonal elements of the original matrix as shown in algorithm 3.

**input :**  $K, \sigma_n$   
**output:**  $L$

$K_n = K + \sigma_n^2 I$   
**while**  $K_n$  not positive definite **do**  
    double  $\sigma_n^2$   
     $K_n = K + \sigma_n^2 I$   
 $L =$  lower Cholesky of  $K_n$

**Algorithm 3:** Lower Cholesky with variance doubling

When doing hyper parameter optimization we do not double the noise variance. Instead our log marginal likelihood function (2.6) returns -infinity for hyper parameters, which produce a non positive definit matrix. Therefore, such hyper parameters do not come into consideration when maximizing.

---

#### 3.2.1 Gaussian Process Regression

---



---

##### 3.2.1.1 Inverse of prior covariance matrix

---

Instead of calculating the inverse of  $K_n$  in (2.2) we use the lower Cholesky decomposed matrix:

$$LL^\top = K_n$$

This is considered faster and numerically more stable [7]. The mean vector  $\mu$  is then computed as follows:

$$\mu = K_n^{-1} y = (L L^\top)^{-1} y = (L^{-\top} L^{-1}) y = L^{-\top} (L^{-1} y) = L^\top \setminus (L \setminus y). \quad (3.3)$$

The backslash operator denotes the matrix left division, so the solution  $x = A \setminus b$  satisfies the system of linear equations  $Ax = b$ . Matrix  $K_n$  must be positive definite for the cholesky decomposition. So we double the noise variance hyperparameter  $\sigma_n^2$  as shown in Algorithm 3.

---

##### 3.2.1.2 Variances

---

For the expected improvement function we only need the vector of variances. Instead of calculating the whole covariance matrix  $V$  and taking the diagonal elements (2.4) we can take a shortcut. All elements on the diagonal of  $K(X_*, X_*)$  equal  $\sigma_f^2$  because the difference between one  $x_*$  and the same  $x_*$  is zero. Therefore we can write:

$$L_k = L \setminus K(X_*, X) \quad (3.4)$$

$$v = \sigma_f^2 - \sum_{\text{rows}} (L_k \circ L_k). \quad (3.5)$$

This adaptation is inspired by [8] and reduces the computational effort drastically from  $\mathcal{O}(n^2)$  to  $\mathcal{O}(n)$ . For the whole covariance matrix for Thompson sampling we also avoid calculating the matrix inverse:

$$V = K(X_*, X_*) - (L_k^\top L_k)^\top$$

WRITE WHOLE MODIFIED GP ALGO BOX

---

### 3.2.2 Action selection

---

In continuous action space our stochastic policy is Gaussian distributed (4.1). Therefore the resulting probability density of the action selection,

$$P_\pi(a|s, x) = \frac{1}{\sqrt{2\pi\epsilon_a^2}} \exp\left(-\frac{(a - f_s(s)x)^2}{2\epsilon_a^2}\right),$$

allows us to do the computations of the logarithm of the probability ratios in the trajectory kernel more efficient:

$$\begin{aligned} \sum_{t=0}^{T-1} \log\left(\frac{P_\pi(a_t|s_t, x_i)}{P_\pi(a_t|s_t, x_j)}\right) &= \sum_{t=0}^{T-1} \log\left(\frac{\frac{1}{\sqrt{2\pi\epsilon_a^2}} \exp\left(-\frac{(a_t - f_s(s_t)x_i)^2}{2\epsilon_a^2}\right)}{\frac{1}{\sqrt{2\pi\epsilon_a^2}} \exp\left(-\frac{(a_t - f_s(s_t)x_j)^2}{2\epsilon_a^2}\right)}\right) \\ &= \sum_{t=0}^{T-1} \log\left(\exp\left(-\frac{(a_t - f_s(s_t)x_i)^2}{2\epsilon_a^2} - \left(-\frac{(a_t - f_s(s_t)x_j)^2}{2\epsilon_a^2}\right)\right)\right) \\ &= \frac{1}{2\epsilon_a^2} \sum_{t=0}^{T-1} ((a_t - f_s(s_t)x_j)^2 - (a_t - f_s(s_t)x_i)^2). \end{aligned}$$

---

### 3.2.3 Hyper parameter optimization

---

Especially for the trajectory kernel we want a hyper parameter optimization, because all the values of the distance matrix  $D$  may get very big. When dividing by a well tuned hyper parameter  $\sigma_l$  before applying the exponential function (2.7), we avoid getting a diagonal matrix  $K$  or a zero matrix  $K_*$ .

When calculating  $\log(|K_n|)$  for the hyperparameter optimization (2.6), again we use the Cholesky decomposition of  $K$ . Thus the determinant transforms to

$$|K_n| = |L L^\top| = |L| |L^\top| = |L| |L| = |L|^2.$$

Since the determinant of the Cholesky decomposed matrix,

$$|L| = \prod_i L_{ii},$$

is the product of its diagonal elements, we can transform this into a numerically more stable version:

$$\log(|K_n|) = \log(|L|^2) = 2 \log(|L|) = 2 \log(\prod_i L_{ii}) = 2 \sum_i \log(L_{ii}).$$

The computation of  $K_y^{-1}y$  in (2.6) is done by the same method we already use in the Gaussian process (3.3).

---

### 3.2.3.1 Independent log-normal prior

---

The log marginal likelihood maximization will sometimes succeed at the borders of our search space resulting in very high or very low hyper parameters. We prevent this by adding an independent log-normal prior term as introduced in [5]:

$$\sum_{i=f,l} \left( \frac{-\log(\sigma_i)^2}{2 \cdot 10^2} - \log 10\sqrt{2\pi} \right).$$

Since the suggested prior is centered at the point of origin with standard deviation 10, but we want our search space to adapt between iterations, we make the mean and the standard deviation adjustable. We write:

$$\sum_{i=1,2} \left( \frac{-(h_i - c_i)^2}{2(b_{u_i} - b_{l_i})^2} - \log(b_{u_i} - b_{l_i})\sqrt{2\pi} \right),$$

where  $b_l$  and  $b_u$  denote the lower and upper bounds of the search space and  $c$  its center. And since we optimize over the logarithmic space  $h_1 = \log \sigma_f$  and  $h_2 = \log \sigma_l$ .

—write whole hyper opt with L and stuff

---

## 4 Experiments

---

### 4.1 Implementation

---

---

#### 4.1.1 Efficiency

---

To cut the time for experiment runs we vectorized every suited operation to speed up calculations in Matlab. Additionally we tuned the code to run on a parallel pool.

---

#### 4.1.2 Global optimization

---

The expected improvement function and the maximization of the log marginal likelihood function for hyper parameter optimization need to be maximized. Since we use MATLAB's GlobalSearch Toolbox, which is a global minimizer, we multiply the results of the functions by -1 to be able to minimize them appropriately.

---

#### 4.1.3 Optimization starting point

---

When optimizing the log marginal likelihood of the gaussian process we are confronted with undefined areas in the hyper parameter space because  $K$  is not positive semi definit in those areas. To start the optimization properly we randomly select points until finding a defined one.

---

#### 4.1.4 Problem environment

---

We use MATLAB implementations of Cart Pole, Acrobot, and Mountain Car that were adapted from [9]. We also implemented the OpenAI Gym to provide a wide range of problem environments. To use the simulations provided by the OpenAI Gym we prepare a python module which is imported to MATLAB. After loading the python module with `py.importlib.import_module(moduleName)` we can call every method it contains via the `py.moduleName.` prefix. It is vital to convert our data correctly before calling the python subroutine with it. The Matlab variables containing whole numbers are converted from Double to Int. Also all the vectors received by the python module have to be converted to an array through `numpy.asarray()`. Unfortunately, the OpenAI Gym environments do not work if we use the parallelization of MATLAB. So we used these environments only with the less computationally demanding local Bayesian optimization.

---

#### 4.1.5 Action selection

---

In continuous action space we use a linear policy to action mapping

$$a = f_s(s)^\top x + \epsilon_a, \quad (4.1)$$

with a small gaussian noise  $\epsilon_a$  needed for stochastic policies. So the actions are Gaussian distributed:

$$a \sim \mathcal{N}(f_s(s)x, \epsilon_a^2).$$

In discrete action space environments we use a parametric soft-max action selection policy:

$$P(a|s, x) = \frac{\exp(f_s(s)^\top x_a)}{\sum_{a \in A} \exp(f_s(s)^\top x_a)}. \quad (4.2)$$

Again it consists of the linear mapping  $f_s(s)^\top x$ , and  $A$  holds all possible actions. The resulting action is then sampled from the probability of action  $a$  given state  $s$ .

The state feature function  $f_s(s)$  depends on the environment as listed in Table 4.1.

Environment	Description	Action	State feature	Policy dimension
Cart Pole	$p$ : cart position $\theta$ : pole angle	continuous: $[-1,1]$	$(p, \dot{p}, \theta, \dot{\theta})$	4
		discrete: $\{-1,1\}$	$(p, \dot{p}, \theta, \dot{\theta}, 1)$	10
Acrobot	$\theta_1$ : angle of pendulum 1 $\theta_2$ : angle between pendulum 1 and 2	discrete: $\{-1,0,1\}$	$(\cos(\theta_1), \sin(\theta_1), \cos(\theta_2), \sin(\theta_2), \dot{\theta}_1, \dot{\theta}_2, 1)$	21
Mountain Car	$p$ : horizontal car position $u$ : car velocity	continuous: $[-1,1]$	$(p, u, p^2, u^2, pu, p^2u, pu^2, p^3, u^3, 1)$	10
		discrete: $\{-1,0,1\}$	$(p, u, p^2, u^2, pu, p^2u, pu^2, p^3, u^3, 1)$	30

**Table 4.1.:** Environment parameters of the experiments

## 4.2 Start conditions

### 4.2.0.1 Global context

We select the initial samples set from a bunch of sample sets such that the Euclidean distance between points of the selected set is maximized. This grants us a well covered search space as a starting condition. BO : We set the initial sample count  $M = 10$  and the iteration count  $N = 200$ .

### 4.2.0.2 Local context

- standardize objective values for acquisition functions Before doing regression we transform our observations to zero mean and uniform variance:

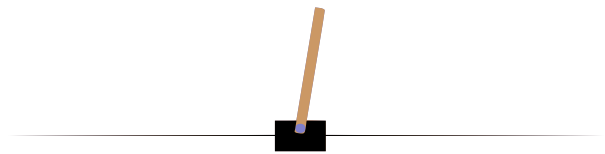
$$y = \frac{y_n - \text{mean}(y_n)}{\text{std}(y_n)}.$$

with a maximum of 500 states

## 4.3 Environments

### 4.3.1 Cart Pole

The Cart Pole environment consists of a cart with a pole attached to its top. The cart is accelerated to the left or the right to balance the pole for as long as possible. An episode starts with small random state values and it ends when the maximum of time steps or a specific state is reached. The state values contain the position and the velocity of the cart, and the angle and the angular velocity of the pole. Each of these four state values are set uniformly random between -0.05 and 0.05 at the beginning. The episode ends after 200 time steps or if the pole falls below 12 degrees relative to the vertical. It also ends when the cart is more than 2.4 meters away from the center. For each completed time step a reward of 1 is returned. The cart pole implementation from Open AI Gym (Figure 4.1) accepts a discrete action value, -1 or 1, for either applying force to the left or to the right. With the four state values and an additional bias value we have five dimensions for each discrete action resulting in a total of ten dimensions. In our own cart pole simulation we only use the four state parameters for calculating an continuous action value between -1 and 1. It turned out that adding a bias value would make the policy search slightly more difficult here. Therefore this policy has only four dimensions.



**Figure 4.1.:** Visualization of the Cart Pole rendered by the Open AI Gym

---

### 4.3.2 Acrobot

---

The acrobatic robot has two joints and two pendulums. The upper joint has a fixed position. It connects to the first pendulum, which is attached to the second joint. A torque can be applied to that second joint, which controls the second pendulum. The pendulums start in equilibrium position and the goal is to gain enough momentum and swing the end of the second pendulum above a certain mark. The angle between vertical and the first pendulum and the angle between the two pendulums form the six state parameters. Four parameters are the sine and the cosine of these two angles each, whereas the last two parameters are the angular velocities 4.1. The discrete action can have the values -1, 0 or 1. Each value stands for the amount of torque applied to the joint between the two pendulum links. With the six state values, an additional bias value, and three discrete action possibilities we get 21 dimensions for the Acrobot policy.



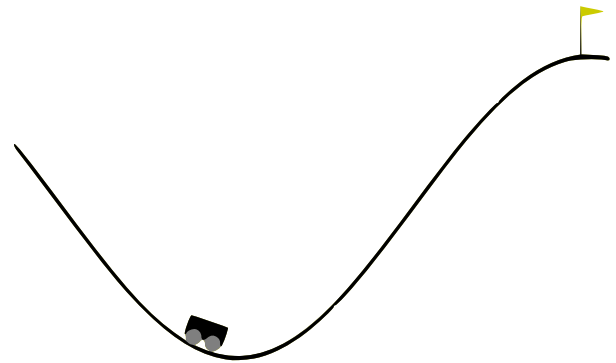
**Figure 4.2.:** Visualization of the Acrobot rendered by the Open AI Gym

---

### 4.3.3 Mountain car

---

During the mountain car task, an underpowered car tries to drive uphill. It can only reach the goal on the right side if it uses both hills to gain momentum. The end of the left hill is an inelastic wall, so if the car hits it, the velocity is set to zero. Since the hills are formed by a sine curve, it is sufficient that the position of the car is only one-dimensional. The vertical position of the vehicle can be easily derived from the horizontal position with the sine function. The horizontal position and the car's velocity are cubically expanded (Table 4.1) to get nine state values. Also a bias parameter is added as a tenth state value. In the continuous setting, we therefore receive ten dimensions. In the discrete setting we have three action values for either applying acceleration to the left or right (-1,1) or doing nothing (0). In this case we receive 30 dimensions.



**Figure 4.3.:** Visualization of the mountain car rendered by the Open AI Gym

---

## 5 Results

TODO

- moving mean 5 - stdv / 5
- debug plotting – mixed up kernel metrics – acq func plots – hyper opt plots — select figure by name

---

## 6 Discussion

TODO



---

## 7 Outlook

- more than one dimensional actions - non-linear action selection policies - higher dimensional environment with traj kernel - measure trajectory differences also with state values, not only action values - Implementing the OpenAI Gym grants us the classic control problems like cart pole, mountain car, and acrobot. Furthermore a lot of more complex environments like a humanoid walker or some Atari games are provided[10]. This will facilitate future research working with higher dimensional problems.

---

## Bibliography

- [1] R. Akrou, D. Sorokin, J. Peters, G. Neumann, *et al.*, “Local bayesian optimization of motor skills,” 2017.
- [2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.
- [3] 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,” *arXiv preprint arXiv:1012.2599*, 2010.
- [4] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, “Taking the human out of the loop: A review of bayesian optimization,” *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.
- [5] D. J. Lizotte, *Practical bayesian optimization*. University of Alberta, 2008.
- [6] A. Wilson, A. Fern, and P. Tadepalli, “Using trajectory data to improve bayesian optimization for reinforcement learning,” *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 253–282, 2014.
- [7] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*, vol. 1. MIT press Cambridge, 2006.
- [8] N. de Freitas, “Python demo code for gp regression,” 2013.
- [9] J. Antonio Martin H., “Matlab sarsa implementation.”
- [10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *CoRR*, vol. abs/1606.01540, 2016.



---

## A Some Appendix

Use letters instead of numbers for the chapters.