Modeling Robustness for Multi-Objective Optimization

Modellieren von Robustheit fuer Pareto-Optimierung

Bachelor-Thesis von Felix Tobias Unverzagt aus Muenchen

April 2016

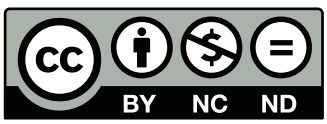
TECHNISCHE
UNIVERSITÄT
DARMSTADT

Modeling Robustness for Multi-Objective Optimization
Modellieren von Robustheit fuer Pareto-Optimierung

Vorgelegte Bachelor-Thesis von Felix Tobias Unverzagt aus Muenchen

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: M. Sc. Roberto Calandra

Tag der Einreichung:



For a lot of awesome robots

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.

Darmstadt, den 27. April 2016

(Felix Tobias Unverzagt)

Thesis Statement

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

Darmstadt, April 27, 2016

(Felix Tobias Unverzagt)

Abstract

Robotic system are often given a high level task, which often consists of several low level tasks which can be optimized. Optimization tasks, such as the energy consumption of a system, can be described as an unknown objective function, which depends on parameters of the system. When optimizing it is of importance that the objective function performs consistently, which requires knowledge about the robustness of the system, usually depending on noise and parameter uncertainty. Furthermore, it can happen that several conflicting optimization tasks such as moving at a fixed velocity and minimizing the energy consumption, have to be performed simultaneously. Such a task can be formalized as a multi-objective optimization problem. Adding robustness as an objective function in multi-objective optimization, provides additional information during the design phase.

Since robotic systems are expensive to run, in this thesis we choose Bayesian optimization (BO) as a main tool. BO learns the minimum of the unknown function of the task, while keeping evaluations on the real system as low as possible. When performing Bayesian optimization, robustness can be estimated through the created model of the system. The estimated robustness can then be used in multi-objective optimization, to increase the knowledge of the system.

To accurately model noise on the system, heteroscedastic Gaussian processes were included, which, in comparison to standard Gaussian processes, are able to model input-dependent noise. Additionally, we propose a new approach to improve the performance of the existing MOO acquisition function EIHV. All algorithms and methods above, especially several implementations for robustness and EIHV, were implemented into a Python framework.

Contents

Figures and Tables

List of Figures

1 Introduction

Many problems in robotics require the optimization of a certain objective, such as the walking speed of a system in locomotion. Often, a model for such an objective can be calculated analytically out of the systems properties and settings. However, analytical calculations are often not derivable, due to the complexity of the problem or unknown system properties. In this calse, one may choose to use black box optimization. In black box optimization a model learns from the system to approximate the objective. This approach is executed by evaluating the objective for certain configurations on the real system, whereas the settings are the input for the model and the value of the objective is the output. As more evaluations are done, the model will fit the real system more accurately and the global minimum of the model will then approximate the global minimum of the objective, which can be seen as the optimal configuration.

Some problems in robotics do not only require the optimization of a singular objective, but of several at once, which is then called multi objective optimization (MOO). A tempting, because simple, solution to solve MOO is to take a linear combination between all objectives and optimize the resulting one dimensional function. The approach of a scalar combination limits the solution to a single point, depending on the preferences which were implied through the weighting of each objective. As the scalar combination is too limiting, it is common practice in MOO to treat every objective independent of the others. Since the former definition of an optimum is the minimum of a single objective and with several objectives usually no global minimum exists, but rather points where changing parameters improves an objective and worsen another, whereas such a point is called Pareto optimal. In this context a Pareto optimal point is a point which can not improve in one objective without getting worse in at least another objective. The set of all Pareto optimal points is called Pareto front, while all Pareto optimal points from a limited sample size are called Pareto set. Using the Pareto set, trade offs between the objectives can be chosen depending on the situations need, as shown in [3].

A problem which often arises, is that samples of the robotic system are expensive to obtain. They can be expensive in regard to time, computational resources or financial cost, whereas expensive in regard to time is true in most of the cases, independent of the other expenses. As resources are not unlimited, the evaluations on the live system have to be limited or possibly minimized, through each new evaluation complementing the already existing evaluations. The so called expensive MOO case has the goal to reduce the number of experiments required to find the Pareto set of the objectives. Usual methods as evolutionary algorithms or gradient descent are too expensive to use, as they are only designed to find the minimum as fast as possible, while assuming almost free system evaluations. Where most algorithm only have a short amount of time, usually milliseconds, to choose the next evaluation, algorithms for expensive MOOs are assumed to have no constrains, while having to return more valuable results with fewer evaluations. To find the parameters for the position where an evaluation will yield the most amount of information, considering earlier evaluations, there exists the definition of the expected improvement, which for each point returns the expectation of how much the model can be improved, by evaluating at that point, but which only can be calculated for models with mean and variance. By optimizing this so called acquisition function, the point for which the most improvement is expected can be selected and then evaluated.
The whole process of choosing a new point to evaluate through the acquisition function, training the model and sampling the model is called Bayesian optimization. While the normal Bayesian optimization and function is only defined for singular objective applications, it is also possible to calculate the expected improvement for the Pareto front, but the definition of improvement changes. Improvement now means, that the space which is bound by the Pareto Set increases [4]. By optimizing the function for the expected improvement of the hypervolume the point which improves the Pareto set the most is calculated.

The goal of this thesis is to provide a framework to calculate robustness for multiple objectives, especially for robotic systems, and to deliver the possibility to use robustness as an objective. Therefore it features Bayesian optimization for multiple objectives through the use of expected improvement of hypervolume [4]. Since robustness computation needs all noises of the system modeled correctly, heteroscedastic Gaussian processes were implemented. To find optimal trade off solutions, the learned models can be sampled to create a Pareto Set. This Pareto set can be expanded through adding the calculated robustness. The contribution of this thesis is the combination of already existing methods, including Gaussian processes, heteroscedastic Gaussian processes, Bayesian optimization and expected improvement of hypervolume, their implementation into a python framework, several different robustness computations and improvement of Bayesian optimization with [4].

Calculating the Pareto set needs a lot of samples, which requires a model to draw data from. The black box optimization model which was used in this thesis is a Gaussian process model. Since our problems have a low amount of input dimensions, as well as continuousness of the input and output, we chose Gaussian process models. Gaussian process models are a commonly and successfully used method for modeling such low-dimensional continuous problems and were implemented through the framework [5]. The hyper-parameters of the Gaussian process models can be optimized by performing gradient descent with random restarts to prevent the optimization from running into a local optima..

Apart from optimizing a certain objective, another goal often arises when handling robots, namely robustness. Robustness is in general a measurement, for how strong the system reacts to any changes, i.e. the more robust the model is the less it reacts to changes. In this thesis robustness is restrained to only certain changes such as robustness to the changes in the parameters of the system $f(x+\varepsilon)$ and robustness of the system, if the parameters do not change $f(x)+\varepsilon(x)$. The paper [2] shows an approach, which models robustness for a fixed volume, including both mentioned possible robustness calculations. If robustness were to be used as an objective it could augment the Pareto Set, thus adding more solutions from which can be chosen, depending on the problem.

The second robustness measure requires a model, which is able to describe different noise on the output function $f(x)+\varepsilon(x)$. While a standard Gaussian process is unable to process more than a constant noise, a Heteroscedastic Gaussian process is made specifically for dealing with changing noises. Several papers dealt with solving the computation of Heteroscedastic Gaussian processes such as [6] and [7]. This thesis uses [7], as it is a recent development for Heteroscedastic Gaussian processes and also comes with a framework, which was also partially used to compute Heteroscedastic Gaussian processes.

In the experiments several different robustness measures were shown and compared and it was tested if Bayesian optimization could be improved through improving the approach of expected improvement of hypervolume [4] through adapting it to the idea from [2], that the Pareto Set of the model is more accurate than the one from the observations. The Experiments showed, that robustness has to be chosen depending on the situation, for example robustness for the worst case or average robustness. Furthermore we show that [4] is performing better in the long run, if the Pareto Set of the model is used in the calculation. Other observations suggest, that using the Pareto Set of the observations for the first 15-25 iterations and then switching to the Pareto Set of the model yields the best results. Additionally, if a function with high noise needs to be approximated, the experiments suggest, that first using a standard Gaussian Process and then switching to a heteroscedastic Gaussian Process yields the best results.

The remaining of this thesis is organized as followed: Section 2 features the mathematical foundations, namely the Gaussian process, Heteroscedastic Gaussian process, Bayesian optimization and MOO. Section 3 includes experiments and their evaluations. These experiments consist out of different approaches to robustness, and analyzing and improving expected improvement of hypervolume in Bayesian optimization. Finally Section 4 concludes this thesis, recapping the contribution of this thesis and the results, as well as showing possible future works.

2 Foundations

2.1 Gaussian Processes

A Gaussian Process (GP) model is a regression model, which is especially good when modeling relations between an input and an output, if their space is continuous and has a low amount of input dimensions. A regression model is a model which maps an input, called independent variables, together with some unknown parameters to an output, called the dependent variable, thus establishing a relationship between them. Note that one model maps to only one output dimension, which is why one model is needed for every output dimension. Supervised learning is roughly learning a function from labeled training data where the function should yield reasonable results on new data. An example for a function which yields unreasonable results on new data is to create a function which fits exactly through several points by having the same polynomial degree as the amount of data, also called overfitting.

Labeled training data is described as (X, Y), with X being the input and Y the output. D is the amount of input dimension and n the length of the labeled training data array. As such it has the dimension $n \times (D+1)$, with each line first containing the input dimensions, labeled with x, followed by the output which is labeled with y. The function which is to be learned will be labeled with $f(x)$. Additionally the input x is transformed into feature space $\phi(x)$, which often helps finding a relationship between X and Y as also shown in [8]. A standard regression model, which is linear in features, with additive noise is now described as followed

$$f(x) = \phi(x)^T \theta \tag{2.1}$$

$$y = f(x) + \varepsilon\,. \tag{2.2}$$

θ is the array of parameters from the linear model, also called weights. Furthermore noise in form of ε was added to the model. This noise is defined as an independent normally distributed noise with zero mean and variance σ^2

$$\varepsilon = \mathcal{N}(0, \sigma^2)\,. \tag{2.3}$$

Bayesian inference can then be used on the model to update the parameters, making use of the Bayes' rule

$$p(A|B) = \frac{p(B|A) \cdot p(A)}{p(B)}\,. \tag{2.4}$$

The Bayes' rule gives way to calculate the possibility that event A is observed, when it is known that event B happens, which is also called the posterior. In other words all the knowledge from event B is taken into account within the posterior. The likelihood is the probability of observing event B after event A happens. By multiplying the likelihood with the previous estimate of the probability that the hypothesis, that event A happens, which is called the prior, the probability for both events A and B happening together is gained. The posterior then is calculated by dividing through the possibility for observing event B without taking event A into account, which is called the marginal likelihood. It is a marginalization as it is usually gained by integrating the likelihood over event A, thus making it independent of A. The now defined Bayes' rule results to

$$\text{posterior} = \frac{\text{likelihood} \cdot \text{prior}}{\text{marginal likelihood}}\,. \tag{2.5}$$

As knowledge is to be gained about the assumption of the parameters θ, the assumption is event A. The other event B is then the labeled training data, which is split into X and Y, which leads to having three events: θ, X and Y. The event X and Y are joined together the event $B = (X, Y)$. Now the formula of Equation (2.4) has to be slightly changed as there are now three events and the observation should have slightly different behavior, than the one gained by simply replacing A and B. As for the likelihood we want to observe the possibility that Y happens depending on X and θ, rather than observing Y and X depending on θ, which is easily understood when having a look at Equation (2.1) and Equation (2.2). As we want to observe such behavior the marginal likelihood also changes to Y depending on X resulting in

$$p(\theta|X, Y) = \frac{p(Y|\theta, X) \cdot p(\theta)}{p(Y|X)}\,. \tag{2.6}$$

In the case of a Gaussian process model all these possibilities are Gaussian distributions. The function described from Equation (2.1) to Equation (2.3) can be rewritten as an equivalent probabilistic function

$$p(y|\phi(x),\theta) = \mathcal{N}\left(y|\phi(x),\theta,\sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\phi(x)^T\theta)^2}{2\sigma^2}\right). \tag{2.7}$$

First of all to calculate the posterior of the now probabilistic function we need the likelihood. With Φ being defined as the vector of all transformed x it can be calculated by factoring over the trainings set because of the independence assumption

$$\begin{aligned} p(Y|\Phi,\theta) &= \prod_{i=1}^{n} p\left(y_i,\phi(x_i),\theta\right) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i-\phi(x_i)^T\theta)^2}{2\sigma^2}\right) \\ &= \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2}|Y-\Sigma\theta|^2\right) = \mathcal{N}(\Phi^T\theta,\sigma^2 I). \end{aligned} \tag{2.8}$$

Next for calculations a prior estimate of the parameters θ is needed. For this purpose a zero mean Gaussian distribution with a scalar Matrix λI is can be used

$$p(\theta) = \mathcal{N}(\theta|0,\lambda I). \tag{2.9}$$

Lastly to calculate the posterior the marginal likelihood is needed. As said above this is gained by integrating the weight out of the likelihood

$$p(Y|\Phi) = \int \mathcal{N}\left(Y|\Phi^T\theta,\sigma^2 I\right)\mathcal{N}(\theta|0,\lambda I)\,d\theta = \mathcal{N}\left(y|0,\sigma^2 I+\lambda\Phi\Phi^T\right). \tag{2.10}$$

With the given equations for likelihood prior and marginal likelihood the Bayes' rule can be used to calculate the posterior which captures all our knowledge about the parameters θ gained from the trainings data

$$\begin{aligned} p(\theta|X,Y) &= \frac{p(Y|\theta,X)\cdot p(\theta)}{p(Y|X)} = \frac{\mathcal{N}\left(\Phi^T\theta,\sigma^2 I\right)\cdot\mathcal{N}(\theta|0,\lambda I)}{\mathcal{N}\left(y|0,\sigma^2 I+\lambda^{-1}\Phi\Phi^T\right)} \\ &= \mathcal{N}\left(\theta|\left(\Phi^T\Phi+\sigma^2\lambda I\right)^{-1}\Phi^T Y,\left(\Phi^T\Phi+\sigma^2\lambda I\right)^{-1}\right) = \mathcal{N}\left(\theta,\mu_N,\Sigma_N\right), \end{aligned} \tag{2.11}$$

whereas $\mu_N = \Sigma_N\Phi^T Y$ is the mean of the Gaussian distribution and $\Sigma_N = (\Phi^T\Phi+\sigma^2\lambda I)^{-1}$ the covariance matrix.
Now the main purpose of a model is to predict behavior for any possible input. For any test point x^* the model should predict the value of the actual function at this point. The function, which the algorithm learned through the trainings data will be labeled with $f^*(x)$. As the probabilities are distributions we are searching for the distribution where $f^*(x)$ is dependent on x^* and also on the trainings data X and Y. Thus we can achieve, that all the information from the trainings data is taken into account. As the information from the trainings data is within the posterior the searched distribution is calculated by averaging over all the possible parameters which are weighted by their posterior

$$P\left(f^*(x)|x^*,X,Y\right) = \int P\left(f^*(x)|x^*,\theta\right)\cdot P(\theta|X,Y)\mathrm{d}\theta = \mathcal{N}\left(\mu(x^*),\sigma^2(x^*)\right), \tag{2.12}$$

which again results in a Gaussian process where the predictive mean $\mu(x^*)$ and variance $\sigma^2(x^*)$ are defined as

$$\begin{aligned} \mu\left(x^*\right) &= \phi\left(x^*\right)^T\lambda\Phi\left(\Phi^T\lambda\Phi+\sigma^2 I\right)^{-1}Y \\ \sigma^2\left(x^*\right) &= \phi\left(x^*\right)^T\lambda\phi(x^*)^T - \phi\left(x^*\right)^T\lambda\Phi\left(\Phi^T\lambda\Phi+\sigma^2 I\right)^{-1}\Phi^T\lambda\phi\left(x^*\right). \end{aligned} \tag{2.13}$$

As calculating now take place in feature space and have dot multiplications the kernel trick can be used. A kernel implicitly computes the scalar product between two sample input points in a higher dimensional space, which is the feature space, without actually needing to transform there. As such all calculations of the dot product in the calculation may be replaced with the kernel defined by $k(x_1,x_2) \sim \phi(x_1)\lambda\phi(x_2)^T$. An often used kernel is the Radial Basis function(RBF) kernel defined by

$$k(x_1,x_2) = \exp(-\alpha\cdot|x_1-x_2|^2), \quad \alpha > 0. \tag{2.14}$$

This is the same as an RBF feature for every position c where the position was integrated out. Furthermore it fulfills two more requirements made to kernels which are them being positive definite and symmetric as they are measuring the correlation between two points. With the definition of the kernel the Equation (2.13) changes to

$$\begin{aligned} \mu(x^*) &= k(x^*, X)\left(k(X,X)+\sigma^2 I\right)^{-1} Y \\ \sigma^2(x^*) &= k(x^*, x^*) - k(x^*, X)\left(k(X,X)+\sigma^2 I\right)^{-1} k(X, x^*)\,. \end{aligned} \tag{2.15}$$

Furthermore the parameters of Gaussian processes can be tuned to match the training data by minimizing the log marginal likelihood

$$log\ (p(Y|X)) = -\frac{1}{2}Y^T(k(X,X)+\sigma^2 I)Y - \frac{1}{2}log|k(X,X)+\sigma^2 I| - \frac{n}{2}log(2\pi)\,. \tag{2.16}$$

Now for the definition of the Gaussian process itself we have to take a look back at the marginal likelihood in Equation (2.10). In this formula if $\lambda^{-1}\Phi\Phi^T$ is replaced with matrix K whereas the entries $K_{i,j} = \lambda\phi(x_i)\phi(x_j) = k(x_i, x_j)$ are scalar products in feature space we yield a $GP(0, K)$. Another form of writing is that a Gaussian prior was placed on the function $f(x)$ resulting in $GP(0, k(x, x'))$. In a lot of approaches, as well as this explanation the base mean μ of the Gaussian Process is set to zero for simplicity. What is not mentioned in both is that we also have a noise term defined in Equation (2.3) which also influences the Gaussian process. We simply assumed there that only one variance exists for the whole noise which is not necessarily true. For this reason we use the heteroscedastic model introduced in the next section. For further questions and more detailed insight into Gaussian processes refer to [9].

2.2 Heteroscedastic Gaussian Process

The Gaussian Process described in Chapter 2.1 has the assumption that the noise on the function is constant as seen in Equation (2.3). The assumption of a constant noise means that if the Gaussian Process is fully trained and has enough data points it will display the same variance for each data point, which is the same as the uncertainty of the model for an output y when the input x is not changing. When the Gaussian process On the other hand if the Gaussian Process is not fully trained it will show its changing uncertainty for an output y when input x is changing due to the model not having enough information about that area. Hereby two noises on the output y can be identified, which are important for a Gaussian process, first $y = f(x + \varepsilon_1)$ which is dependent on the accuracy of the model and second $y = f(x) + \varepsilon_2$ which is the actual noise on the function and which is assumed to be constant.

While assuming a constant noise yields good results, if the real noise is close to being constant itself, the results get worse the more the real noise on the function is behaving non constant. To picture it more clearly one can imagine making a call while driving a car, where the signal is more or less blocked depending on the environment which can lead to extreme noise differences. Also the noise of a system can change when the forces in the system change, which they do for example while lifting an arm. While there was only the gravitational force before lifting there is now an additional torque, which might lead to a stability change within the system which influences the noise. Also when the model takes time into account it cannot take resonance disasters or resonance at all into account.

Thus we need a model which is able to also change the variance depending on the position in input space. Such a model is the Heteroscedastic Gaussian process as described in [7]. We chose to take [7], as it is a recent work in the area of Heteroscedastic Gaussian processes and provides an out of the box working framework in combination to a sophisticated approach. They describe their variance by first using a Gaussian Process by placing a Gaussian prior which they paremetrize as seen in Equation (2.17) to ensure positivity of the variance. Their equations written in the style of this paper are defined as

$$\begin{aligned} f(x) &\sim GP\left(0, k_f\left(x, x^T\right)\right), \quad \epsilon \sim \mathcal{N}(0, r(x_i)) \\ r(x) &= \exp(g(x)), \quad g(x) \sim GP\left(\mu_0, k_g\left(x, x^T\right)\right). \end{aligned} \tag{2.17}$$

Notice that we have two Gaussian Processes now, one for the variance and one for the function. This is no longer analytically tractable, but can be made to be so if some assumptions are made as described in [7]. They [7] also show how the expected mean $\mathbb{E}(x)$ and the variance for the Heteroscedastic Gaussian process are calculated from the kernels for any input x^*

$$\begin{aligned}\mathbb{E}(x^*) &= \mu_1(x^*) \\ \mathbb{V}(x^*) &= \sigma_1^2(x^*) + \exp\left(\mu_2(x^*) + \sigma_2^2(x^*)/2\right),\end{aligned} \tag{2.18}$$

whereas μ_1 is the mean calculated from kernel from the first Gaussian process, which is also called mean kernel and σ_1^2 is the variance calculated from the mean kernel. μ_2 is the mean calculated from the variance kernel, which is the kernel from the second Gaussian process and σ_2^2 is the variance calculated from the variance kernel

$$\begin{aligned}\mu_1(x^*) &= k_f(x^*,X)(k_f(X,X)+R)^{-1}Y \\ \sigma_1^2(x^*) &= k_f(x^*,x^*) - k_f(x^*,X)(k_f(X,X)+R)^{-1}k_f(X,x^*) \\ \mu_2(x^*) &= k_g(x^*,X)^T(\Lambda - \frac{1}{2}I) + \mu_0 \\ \sigma_2^2(x^*) &= k_g(x^*,x^*) - k_g(x^*,X)^T(k_g(X,X)+\Lambda^{-1})^{-1}k_g(X,x^*).\end{aligned} \tag{2.19}$$

Λ is a positive semidefinite diagonal matrix, whose actual value is calculated while optimizing, and R is also a diagonal matrix, whose elements calculate from the mean and covariance values of the normal distribution used in [7] to compute the Marginalized Variational bound. The log marginal likelihood which is minimized to tune the parameters is defined by

$$log\ (p(Y|X)) = log\left(\mathcal{N}\left(y|0,k_f(X,X)+R\right)\right) - KL\left(N\left(g(X)|\mu,\sigma\right)||N\left(g(X)|0,k_g(X,X)\right)\right) - 0.25tr(\Sigma). \tag{2.20}$$

KL is the Kullback–Leibler divergence and tr the trace, while Σ holds the covariance matrix of the normal distribution used in [7] to compute the Marginalized Variational bound. Due to more complex calculations and essentially two Gaussian Processes within this heteroscedastic Gaussian Process approach the computation time is higher than for a standard Gaussian Process. As the shown equations are all which is needed for this thesis to compare to the standard Gaussian Process, refer to [7] for further insights in heteroscedastic Gaussian Processes.

2.2.1 Example

To show the extreme difference in the models learned, when using a heteroscedastic Gaussian process instead of a traditional Gaussian process, we show a sinus curve $y = sin(4\pi x)$ to which three different true noises ε_t will be added

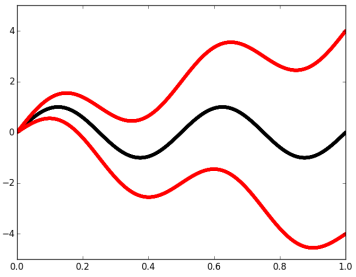
$$\varepsilon_{t_1} = \mathcal{N}(0,2x) \tag{2.21}$$

$$\varepsilon_{t_2} = \mathcal{N}\left(0,(2x)^2\right) \tag{2.22}$$

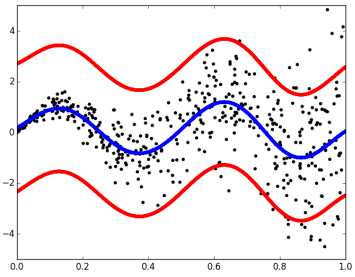
$$\varepsilon_{t_3} = \mathcal{N}\left(0, negToZero\left(sin(3\pi X) + \frac{sin(4\pi X)}{1.5} + \frac{sin(5\pi X)}{2} + \frac{sin(7\pi X)}{3}\right)\right) \tag{2.23}$$

$$negToZero(x) = \text{for } x_i \text{ in } x \text{ if } (x_i < 0)\, x_i = 0.$$

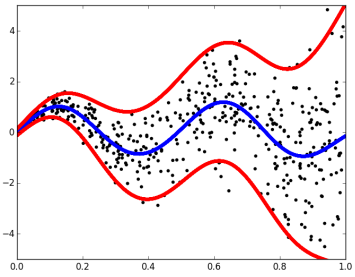
The progress spans from a variance which a normal Gaussian Process can handle quite good over to one it can no longer fit. The models were trained with 500 random sampled points between zero and one. The start was a linear noise, where the mean is learned perfectly from both models, and the variance is not that much of a difference in the Gaussian Process and perfect with the heteroscedastic as seen in Figure 2.1. Next is the same noise, but squared, which makes the normal go way to high with the variance and even the heteroscedastic is not exact towards the end, but that can be explained due to undersampling at this position shown in Figure 2.2. The last example is like the phone call described above or actually any signal which is obscured for some parameters.
To guarantee a function which changes a lot we added several sinus curves with different frequencies on top of each other. As a noise is always positive and we wanted to have some areas where it is zero we then filtered the function by setting every value below zero to zero. Resulting from such an erratic function the Gaussian Process is doing extremely poorly at the beginning and quite bad at the end. Though this is a result from undersampling, as a Gaussian Process will always get the mean right with enough samples and good parameters calculation, from which we only have the last, one sees that 500 random samples should hopefully be enough. The heteroscedastic model manages the mean value perfectly, but due to not enough samples at the first peak its variance there is off by a bit as shown in Figure 2.3.
As shown heteroscedastic Gaussian Process models are able to achieve better representation of systems with changing noise, by having a changing variance themselves and are able to approximate the mean function better with few data points if a lot or erratic noise exists.



(a) sinus curve(black) and 2x deviation of real noise(red)

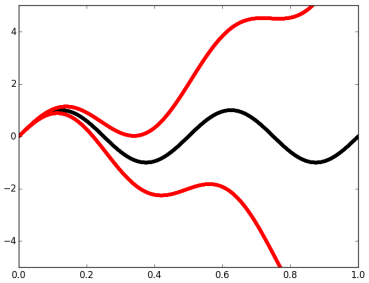


(b) mean(blue), 2x deviation(red) of GP model and trainings points(black)

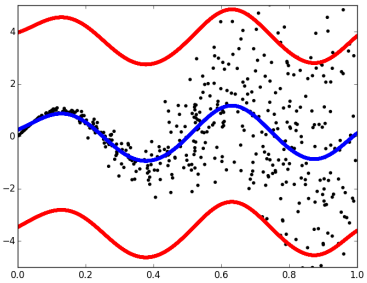


(c) mean(blue), 2x deviation(red) of hetero GP model and trainings points(black)

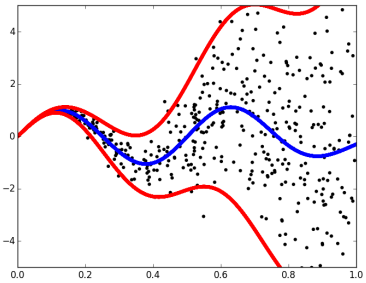
Figure 2.1: Sinus with linear noise function



(a) sinus curve(black) and 2x deviation of real noise(red)



(b) mean(blue), 2x deviation(red) of GP model and trainings points(black)



(c) mean(blue), 2x deviation(red) of hetero GP model and trainings points(black)

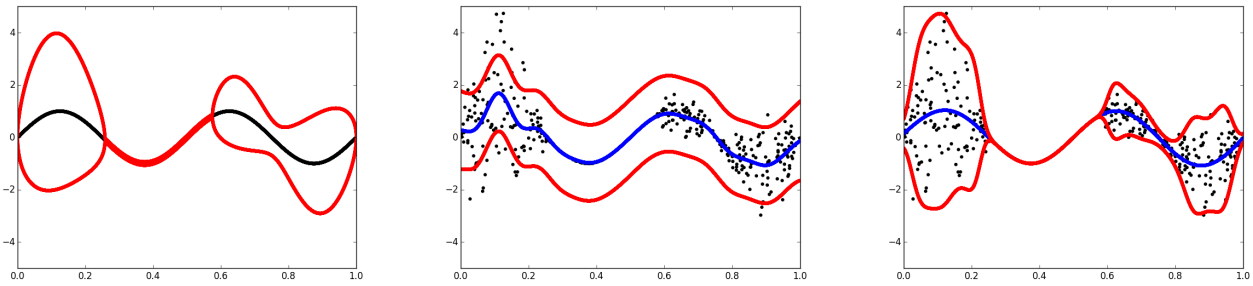
Figure 2.2: Sinus with quadratic noise function

2.3 Bayesian Optimization

Optimization is the process of finding the most optimal state in terms of some predefined objective. Within the scope of this thesis an optimal state shall be found by minimizing an unknown function $f(x)$. As for finding the minimum we are searching for a global minimum within certain borders. The function itself is unknown and we have a Gaussian Process model instead, which is learning it. In particular the standard Gaussian Process, which was described in Section 2.1 is featured and not the heteroscedastic one, although the idea stays the same.

Calculating the minimum of the model at a given state only requires a very fine grid search depending on the accuracy which should be achieved. Providing that the model perfectly shows the behavior of $f(x)$, such a result would be satisfying, but as can be seen in Figure 2.3, this is not the case for the Gaussian Process, even with 500 training points, which already is a lot. Such an amount of training data might prove to be very expensive or even impossible if taken only from real life runs. Furthermore the modeling of the function gets worse the more noise exists. So in order to find a minimum of $f(x)$ the model has to be accurate enough. This accuracy can be described, that for the actual state of the model it has to be sufficient sure, that its minimum is the minimum of the actual function. Such can be provided by showing, that the possibility of the minimum being somewhere else is sufficient small.

Bayesian Optimization offers the calculation of such an accuracy, while also being able to train the model to obtain the accuracy in likely the most efficient way. To get an idea how this works and how a not fully trained Gaussian Process looks like, we defined a function which can be seen in Figure 2.4a, and start with 6 random training samples or observation, seen in Figure 2.4b.
The function itself has a minimum at about (0.437,-1,09), while the model thinks it lies at (0.402,-1,033). The basic idea is to define a function for the model, which describes the improvement of the model towards catching the minimum in some way, if an observation were to be made at a point. This function, which is called acquisition function, is made to be relatively cheap to compute and if it is optimized it returns the point where the model will improve most if an observation is made. In Figure 2.4c the general space where the function would be able to improve is shown. It is to

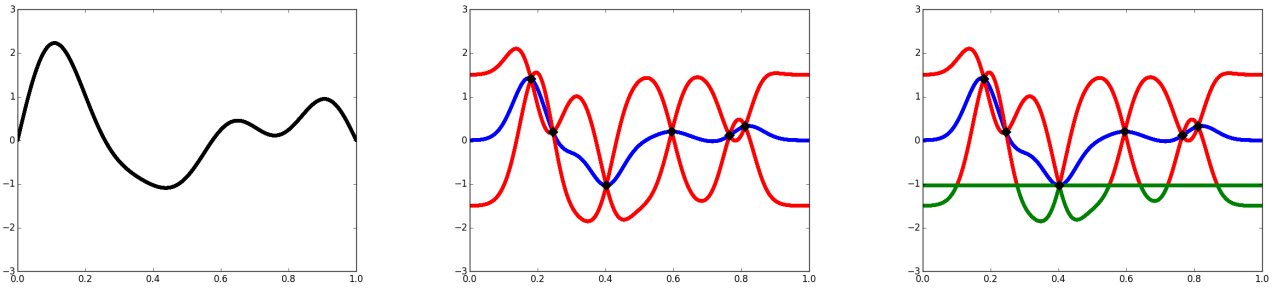


(a) sinus curve(black) and 2x deviation of real noise(red)

(b) mean(blue), 2x deviation(red) of GP model and trainings points(black)

(c) mean(blue), 2x deviation(red) of hetero GP model and trainings points(black)

Figure 2.3: Sinus with irregular noise function



(a) function to learn

(b) Gaussian Process with 6 training points, mean(blue), 2x deviation(red)

(c) mean(blue), 2x deviation(red), space for improvement(green)

Figure 2.4: Gaussian Process modeling a function with 6 trainings points

note, that though the variance is important, it only matters if it the model could think, that a value under the actual minimum could exist there. By this method, places with high uncertainty can be left alone if the minimum could not be there.

There are several different functions, like the expected improvement or the entropy search where the uncertainty in the location of optimal value is minimized. We give a quick insight into the expected improvement method to show how acquisition functions work. To do this we first have to explain what the probability of improvement is. If the minimum of the model is called f^*, improvement takes place everywhere, where the actual function is smaller than this minimum, thus we define a reward function

$$r(x) = \begin{cases} 0 & f(x) > f^* \\ 1 & f(x) \leq f^* \end{cases}, \tag{2.24}$$

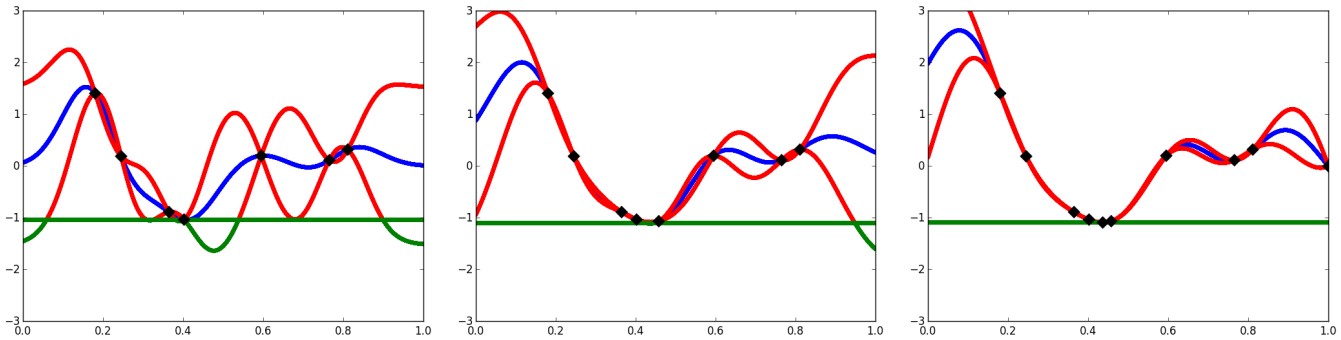
whereas f(x) is the real function which is to be learned. The acquisition function for the probability of improvement is then calculated by taking the expected reward

$$\begin{aligned} A(x) = E[r(x)|x,X,Y] &= \int_{-\inf}^{f^*} \mathcal{N}(f;\mu(x),k(x,x))df \\ &= \Phi(f^*;\mu(x),k(x,x)). \end{aligned} \tag{2.25}$$

This function can be maximized to find the point with the highest possibility for improvement. With the expected improvement the process is quite similar to the probability of improvement, but we have to define a new reward function, which gives the expected improvement

$$r(x) = max(0, f^* - f(x)) \tag{2.26}$$

$$\begin{aligned} A(x) = E[r(x)|x,X,Y] &= \int_{-\inf}^{f^*} (f^* - f)\mathcal{N}(f;\mu(x),k(x,x))df \\ &= (f^* - \mu(x))\Phi(f^*;\mu(x),k(x,x)) + k(x,x)\mathcal{N}(f^*;\mu(x),k(x,x)). \end{aligned} \tag{2.27}$$



(a) seven trainings points **(b)** eight trainings points **(c)** ten trainings points

Figure 2.5: Example of Bayesian optimization, mean(blue), deviation(red), space for improvement(green)

The reward function is positive if the value of the function $f(x)$ turns out to be lower than the minimum f^* and zero otherwise. Expected improvement is split into two parts, the left one being the probability of improvement multiplied with the difference between the minimum and the average value at x. This side gets bigger if $\mu(x)$ gets smaller. The right side is the covariance function, or the kernel, multiplied with the possibility of reaching the minimum at x. This side gets bigger by increasing the value of the kernel for x.

2.3.1 Example

As one acquisition function was described, we will show how the Gaussian process from Figure 2.4 develops with some steps through Bayesian Optimization. The framework ROBO and its implementation of the Expected Improvement and Bayesian Optimization was used to execute the following steps. In Figure 2.5 the seventh, eighth, and tenth step are shown as the difference between nine and ten is almost not visible and the tenth step already modeled the minimum almost perfectly. It could be shown, that through a small amount of steps a minimum can be reached fairly quick. The downside is, that the evaluation of the acquisition function takes some time, but which is almost irrelevant with real life runs as the runs are usually very expensive. Bayesian Optimization provides a quick way to train a Gaussian Process model to find the real global minimum of any function.

2.4 Multi-objective optimization

All previous sections featured only one objective function $f(x)$, with multiple input dimensions x. However in optimization there are usually multiple objectives $F(x)$ and with the idea of introducing robustness as an additional objective function, this paper focuses on at least two objectives. Due to differences while handling several objectives in comparison to handling one objective the changes which have to be made for the previous sections will be explained. First of all either Gaussian process described in Section 2.1 or Section 2.2 is unable to deal with multiple output dimensions, neither is it in the Gaussian Process framework [5], which was used in this paper. This can be solved, by creating one individual Gaussian Process model for each of the output dimensions. Such an approach assumes, that the objectives are independent of each other, or that the dependence can be modeled solely out of the input parameters, thus treating them independent. If an objective, like the robustness, is also directly dependent on other objectives, it will be calculated after the other models are trained, as they are independent of it and the model should then have all information it can get from the input. Such objectives as robustness will not be modeled in this paper, as the dependencies to the input cannot be directly modeled.

2.4.1 Example of an multi objective function

To visualize a multi objective function we chose the MOP2 function [1], which is a standard test function for MOO and thus also used in the experiments

$$
\begin{aligned}
y_1 &= 1 - \exp\left(-\Sigma_i \left(x_i - \frac{1}{\sqrt{2}}\right)^2\right) \\
y_2 &= 1 - \exp\left(-\Sigma_i \left(x_i + \frac{1}{\sqrt{2}}\right)^2\right).
\end{aligned}
\tag{2.28}
$$

We evaluated this function for two input dimensions from the space -2 to 2 in each dimension and Figure 2.6 now shows several possibilities to visualize these multiple objectives. The most important thing to note is that in addition to the parameter space, which was used in all figures before and in Figure 2.6a and Figure 2.6c, on observable n-dimensional objective space exists additionally which can be seen in Figure 2.6d. Such an objective space already exists with only one objective function, but is one dimensional. In such an one dimensional space only the minimum of the function can be observed, which was the lowest value in objective space. In general, also with multiple objectives, a lot of the objective space is empty, as there are usually minimum and maximum objective values which can be achieved within the set borders for the input parameters.

In Figure 2.6a the first objective function is shown in parameter space. Figure 2.6b shows the result of the second objective function also in parameter space, whereas the third Figure 2.6c shows both objectives added together. When Figure 2.6c is transferred into objective space it results in Figure 2.6d. For the last two figures the colors were chosen the same such that it can be seen how the parameter space maps into the objective space. An information which can be gained from this visualization is that the mapping from parameter to objective space is not invertible. Due to the visible symmetry of the function to $x = y$ at least two different input values will result in the same output. The most interesting observation which can be made is the line between the minimums for each objective at $(1,0)$ and $(0,1)$, which can be seen as a trade of between them. This line is called Pareto Front and will be covered in the next section.

2.4.2 Pareto Front

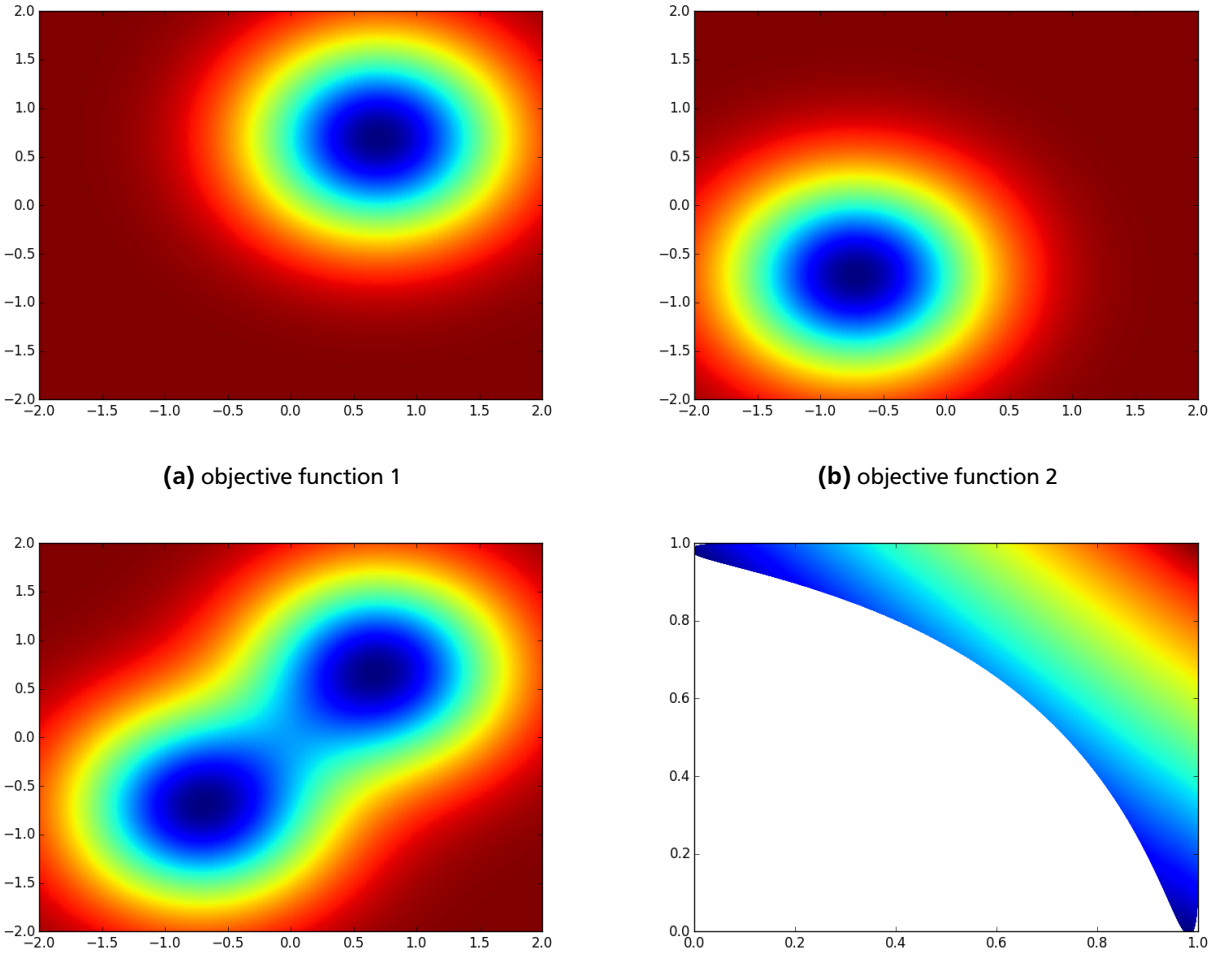
In MOO, the minimum for each objective can be calculated, but as we have several objectives the trade off between their minimums is now interesting. Such trade offs were often calculated by minimizing a function where all objectives were added together, while each objective was weighted depending on the users needs. By doing so the actual problem from MOO was avoided, as it was reduced to single objective optimization. Each trade off had to be handpicked, possibly in many tries and if several trade offs where needed the same had to be done for each. The idea of the Pareto Front provides all possible trade offs, while also possibly providing a mapping from objective to input space. Depending on the function which has to be minimized and on the time which can be spend to evaluate said function, it might actually not be possible to calculate all trade offs.

For the definition of the Pareto Front the definition of a Pareto optimal point is needed. A Pareto optimal point is a point, where an objective cannot be minimized further, if minimizing is the goal, without getting worse at another objective. As such a point $y \in \mathbb{R}^n$, whereas n is the number of objectives, dominates another point y', if $\forall i \in 1,...,n : y_i \leq y'_i$ and $y \neq y'$.
The Pareto Front is defined as the set of all Pareto optimal points. With n objective functions this results in a $\mathbb{R}^n$ dimensional surface, which is limiting the target set. All points which are the minimum for each separate objective are the points which span the plane. Thus the Pareto Front can be seen as a trade off between those minimums, which is also why the grid for the acquisition function, which is a trade off between minimums, is important, as one want to model the Pareto Front as accurate as possible. As the Pareto Front is usually calculated from a set of points we give a quick description of how to check if a y_1 dominates another point y_2

$$
y_1 \prec y_2 \iff \forall i \in 1,..,n : y_{1,i} \leq y_{2,i} \wedge y_1 \neq y_2 \,. \tag{2.29}
$$

This comparison now has to be done for each combination of points, to check if any point is dominating another. Mathematically this means $n + (n-1) + (n-2) + ... + (n-n+1) + (n-n)$ comparisons for n data points, which means a n^2 scaling. When actually deleting non Pareto optimal points while comparing n^2 is only the worst case scenario for all data points being Pareto optimal and an order of n is the best case where one point dominates all others. In real life runs the computation time usually results to something in between n and n^2, although much closer to n than n^2.

To show the looks of a Pareto Front we use again the MOP2 function[1], introduced in the section before. In Figure 2.7a the Pareto Front is shown in input space, while in Figure 2.7b it is shown in objective space. This Pareto Front was



(a) objective function 1

(b) objective function 2

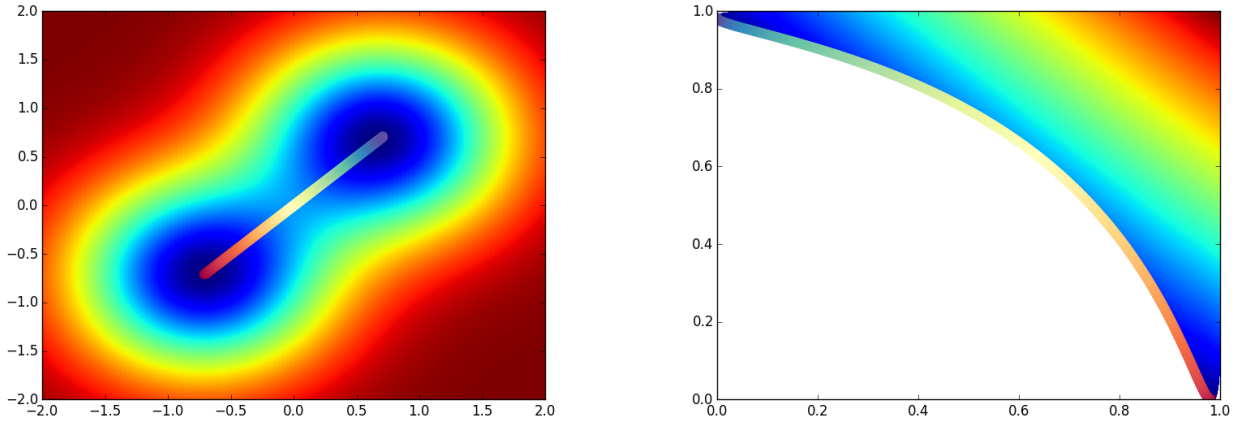
(c) objective 1 + objective 2

(d) objective space(color = obj1+obj2)

Figure 2.6: Three basic visualizations for MOP2 [1] function

gained by creating 25000 points as a grid in input space and then calculating the Pareto optimal points from this set. The amount of Pareto optimal points calculated by this process was 355, which is only 1.42% of the original data set. Although the extreme amount of initial points was done for visual reason, as the pictures where also made with them, it is apparent, that such data cannot be obtained through real life runs, but easily from a model. Especially the fact that only about 1.42% of the data lie on the real Pareto Front means that an average of 70 runs would have to be done to get only one Pareto optimal point. Although the amount of runs would decrease drastically if some sort of predictions would be used, the predictions would just be a replacement for the model. In [2] the difference between a Pareto Front gained from evaluations and and the one gained from a model trained with those evaluations is depicted. Sampling 20, 50, 100, 200 input parameters from a uniform distribution, they evaluated them on the MOP2 [1] function and trained the model with the resulting training data. For 20 uniform sampled training points, the predicted Pareto Front from the data, as well as the one from the model is suboptimal, although the model delivers better results. Starting at 50 data points the model calculates the Pareto Front with only a small error. Meanwhile the calculation from the data is still bad even at 200 evaluations, while the model provides a Pareto Front which can visibly not be differentiated from the real one.

In another example they also show, that the Pareto Front gained from data is wrong, if noise exists. For such a case a Gaussian Process model delivers not only a dense, but also a more correct Pareto Front.
In [3] an actual example for the use of the Pareto Front is described. They feature a robot snake with a camera as the head, with two objectives, speed and visual accuracy from the camera, which are in direct conflict with another. The faster the snake moves, the more it head moves thus making it harder to keep visual control. They then chose three points on the Pareto front showing the different movement. As the definition of the Pareto front goes none dominates another, making each of the settings fit for its own purpose. The first one was chosen to move with a higher speed with unstable visual input, while the last one focused on having a stable picture and the middle one being a trade off between



(a) input space

(b) objective space

Figure 2.7: Pareto Front for the MOP2 [1] function

both. It is again to note, that the Pareto Front provides a lot of trade off solutions at once compared to other approaches to aggregate objectives.

Additionally to the simple mapping from objective to parameter space, by choosing a point in objective space, where the parameters are known, in some cases the Pareto Front offers the possibility of learning how to map from objective to input space. In Figure 2.7 it can be seen from the colors, that a simple mapping from objective to input space is impossible for the whole model, due to the symmetry to $x = y$ in input space. For the Pareto Front this is another case, as it lies on said symmetry axis and therefore a mapping from objective to input space can be learned. It is also not by chance that this is happening as the Pareto Front is a trade off between the minimums of each objective function. As in most mechanical systems movements are not abrupt the Pareto Front is usually a connection between the minimums, as well in input space as in objective space. The properties of this connection then can be learned to map from objective to input space. As indicated above, this is not applicable for all systems, as a lot of complex systems have abrupt changes different local minimums competing with each other, where then the Pareto Front is not easily traceable within the parameter space.

2.4.3 Expected improvement of hypervolume

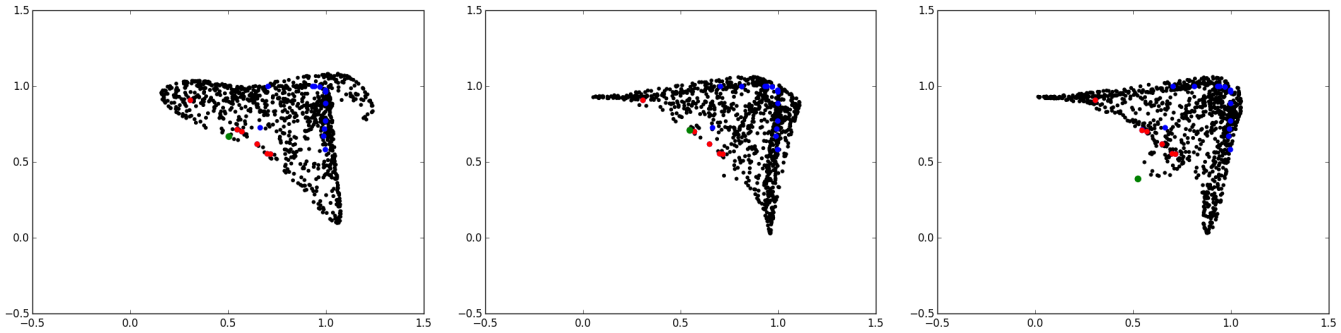
The training of the Gaussian Process models also changes significantly, as the former definition for the minimum of the function $f(x)$ is no longer valid for $F(x)$, as now a Pareto Front in $\mathbb{R}^n$ exists, whereas n is the number of objectives. As such the expected improvement method is no longer usable as defined in Equation (2.25). Improvement can instead be seen as the improvement of trade off minimums, which in turn is the Pareto Front. The Pareto Front is dividing the codomain into two volumes, one in which no point is dominated by the Pareto Front and another in which all points are dominated. The volume in which all points are dominated is defined as the hypervolume bound by the Pareto Front and will be referred to as hypervolume. Improvement of the Pareto Front can now be seen as an increase of the hypervolume. With P defined as an Pareto Set from evaluations, y being a point in the codomain and $H(P)$ the hypervolume of P, the improvement I is defined as

$$I(y,P) = H(P \cup y) - H(P)\,. \tag{2.30}$$

With the probability density function $PDF_x(y)$ which defines the probability of the relative likelihood of x becoming y, the expected improvement of the hypervolume can be defined as the integral over the improvement for all points $\overline{y}$ in $\mathbb{R}^m$ multiplied with their probability depending on x

$$EI(x) = \int_{\overline{y}\in\mathbb{R}^m} I(\overline{y},P)\cdot PDF_x(\overline{y})d\overline{y}\,. \tag{2.31}$$

This integral can be calculated by a Monte-Carlo integration method. But due to this method having limited accuracy and it allowing no direct computation [4] proposes a direct method for computing the expected improvement in the multidimensional space.



(a) 18 training points **(b)** 19 trainings points **(c)** 20 trainings points

Figure 2.8: Bayesian optimization with EIHV, training data(blue), 1000 random model points(black), Pareto set from training data(red)

Their approach is basically to separate the $\mathbb{R}^n$ space into n-dimensional volumes and then sum the contributions of said volumes. The volumes are created by laying a grid through the $\mathbb{R}^n$ space, such that all points of the Pareto Set are grid points. Furthermore to simplify calculations the contribution to the integral is only calculated for volumes where improvement can take place. These volumes are called active and they are all the volumes which are not dominated by the Pareto Set.

Expected Improvement of MOP2 - Example

Unlike in the example for Bayesian optimization seen in Figure 2.5 where the function has one input and one output dimension the MOP2 function described in Equation (2.28) in this case has two input and two output dimensions. As the improvement of the Pareto Front is of interest, the plot will be shown in objective space. In Figure 2.8 the 14th, 15th and 16th iteration of Bayesian Optimization is shown. The green points are the points gained from observations, the red points are the Pareto Set gained from the observations and the black points 1000 random sampled and with the model predicted points. From the figures it is quite clear, that the approach from [4] is working as intended, as from Figure 2.8a to Figure 2.8b an outer point is corrected such, that the model is extremely more accurate and in Figure 2.8c the most outer point is selected as there the most improvement could take place.

3 Experiments

3.1 Experimental Environment

All experiments were written and executed in the programming language python. As basis for the implementation GPy was chosen, which provides several standard Gaussian Process Regression models, as well as different likelihoods, kernels and inference functions. The Gaussian Processes used in the experiments are a standard Gaussian Process with square exponential kernel from [5] and a heteroscedastic Gaussian Process, implemented from [7], also with square exponential kernel. Finally in all the runs when learning a Gaussian Process, gradient descent with 100 random restarts was used if not specified otherwise.

3.2 Exploring Robustness

Robustness is an important issue in optimizing, especially when working with robots. In this case robustness is defined so that a point is robust, if changing the parameters in parameter space a little, results in a small change in objective space. The need to specify either an area around each point in which robustness is measured or how changes are weighted depending on their distance to the point arises.
This section features different trade offs and areas, which will be presented and discussed to explore multiple ways to interpret robustness. Each individual algorithm will then be evaluated and rated. As function to execute and test the algorithms on the MOP2 function described in Equation (2.28) was chosen, which is a standard test function for MOO and was introduced in Section 2.4.
Also the input dimensions were set to two, since two dimension are best for visualization purposes. When using two input dimension there exist two indicators, that the basis of an algorithm is correct, when plotting in parameter space. The first indicator is that since each input dimension is treated the same way, a symmetry to $x_1 = x_2$ exists. This symmetry will always exist, whatever algorithm is used which can be checked by taking any point (a, b) and exchanging the input parameters to (b, a). The second indicator is the sign change before $\frac{1}{\sqrt{2}}$ when comparing y_1 and y_2, which leads to $y_1(a, b) = y_2(-a, -b)$. A symmetry to $x_1 = -x_2$ can then be seen, if both output dimensions are weighed the same way. Furthermore the domain begins at -2 and ends at 2 for both input dimensions.
The summarized knowledge about robustness functions collected so far: The function must define an area around each point from which the robustness is sampled. It must define how it weights the data in this area, e.g. weight changes further away less than those closer. Additionally if used with the MOP2 function, the robustness function has to be symmetric to $x_1 = x_2$ and it has to be symmetric to $x_1 = -x_2$, if both output dimensions are weighed the same way.
To get an idea how a fully developed robustness function may look like the robustness from [2] was chosen. In [2] a fixed square around each point was defined to sample robustness. For the weighting [2] then chose to sample 2000 points of that area and calculated the average of the probability of those values for the normal distribution function of the original point. This value was then divided by the value of three times the standard deviation of the normal distribution of the original point. Their result can be seen in Figure 3.1.
Marked with A and B in this picture are the Pareto optimal points for each output dimension. The small slightly brighter line between those points is the Pareto front. C and D mark the area with the best robustness, which itself is very far away from the Pareto front.
Both symmetries are given in the approach as their robustness computation does not distinguish between the output dimensions. As this result is highly optimal and adaptable, but has the drawback of an immense computation the focus in the following approaches will be on algorithms which have a lower computation time.
In the following the process for developing a good robustness function with different approaches is pictured. The results will be explained on a mathematical basis and evaluated. In the end it shall be shown, that alternatives to [2] can be achieved with lower computation costs and different algorithms. Every algorithm in the following describes what will be done for each point which is to be evaluated. Since writing this each time would be redundant it is just left out. Notice though that the pictures are all made by evaluating each algorithm 10201 times, because a grid with 101 points in each directions was chosen. Additionally the minimum value of the robustness was subtracted and then the value was divided by the maximum value such that the worst robustness is 0 and the best 1.
Furthermore there exist two different types of noises to which an algorithm can be robust. The first is the noise when the parameters of the system change $f(x + \varepsilon)$, where robustness means, that if small changes happen in x only small

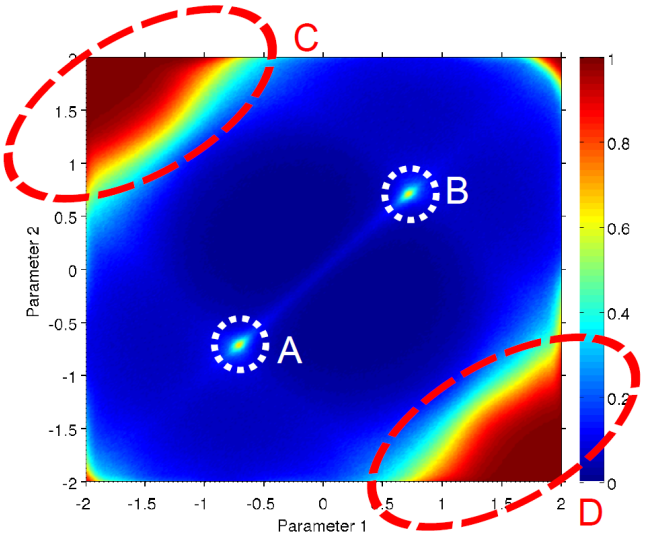


Figure 3.1: Average area Robustness from [2]

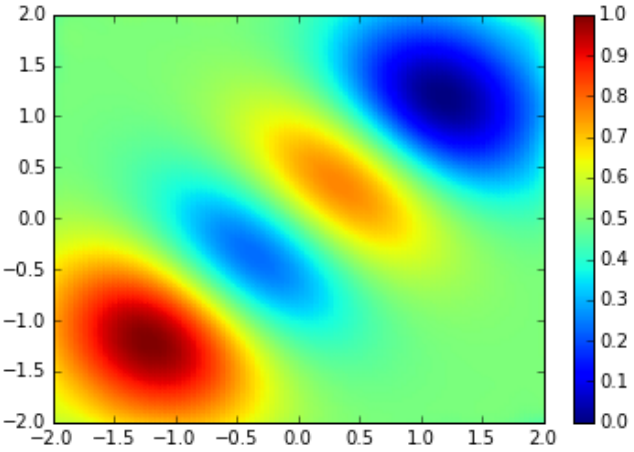


Figure 3.2: Adding all the derivates up.

changes exist in y. The second is noise when the parameters do not change $f(x)+\varepsilon(x)$, where robustness means, that this noise is small. First a series of approaches for robustness when the parameters change $f(x+\varepsilon)$ is shown.

3.2.1 Derivate approach

As a Gaussian Process uses smooth functions to approach the real function a derivate should show a good assumption of the change in a small area around a point. The size of the area in which this approach is always close to being exact depends on the specifications of the Gaussian Process and on the function which is to be learned. Since a derivate approach is basically a Taylor approximation it can be said that for a distance d from the point errors in $O(d)$ are made. As most functions don't behave the same at every point for every input dimension, the resulting area where changes are approximated correctly is most of the time a rectangular, different to the square used in [2]. Finally if one uses only the first derivate, as the following presented algorithms, there is no weighting within the area.
A first naive approach to calculate the robustness is to simply add up all the derivatives of all models for every dimension and then negating the value as seen in Figure 3.2. The assumptions for this approach are that a big derivate results in a bad robustness value and that all input dimension may be treated the same way.

```
robustness = 0;
for outputDim = 1,2...,numOfOutputDims do
    for inputDim = 1,2...,numOfInputDims do
        robustness = robustness + model.getDerivateValue(actualPoint,inputDim,outpuDim);
robustness = -robustness;
```

Algorithm 1: Adding all the derivates up and inverting

Though the necessary symmetry to $x=y$ exists, the symmetry to $x=-y$ is not there, though from the idea both output dimensions have been treated the same way. After some investigation the error was simply that the absolute of the derivatives was not taken, which resulted in not adding the robustness in each dimension but just the derivate together. Thus the next step is to take the absolute value of each derivate before summing them up. This results in a perfect symmetry to $y=-x$, which can be seen in Figure 3.3.

```
robustness = 0;
for outputDim = 1,2...,numOfOutputDims do
    for inputDim = 1,2...,numOfInputDims do
        robustness = robustness + abs(model.getDerivateValue(actualPoint,inputDim,outpuDim));
robustness = -robustness;
```

Algorithm 2: Adding all the derivates absolutes up and inverting

This is as far as this algorithm can go in the direction of [2]. In comparison to Figure 3.1, the Pareto optimal points A and B can be seen, though not the Pareto front itself. Furthermore the trend of being more robust towards the corners of

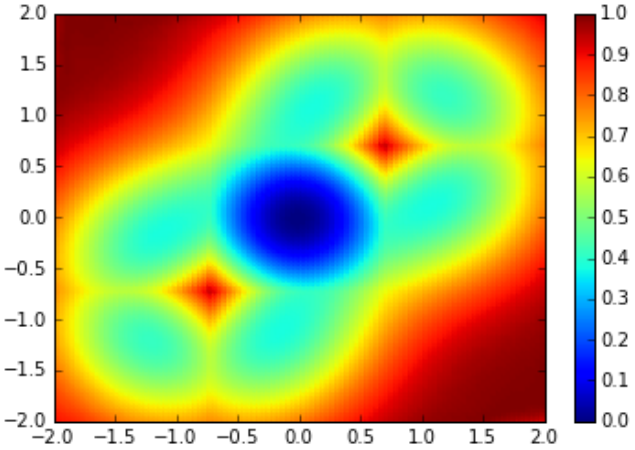


Figure 3.3: Adding all the absolute derivatives up.

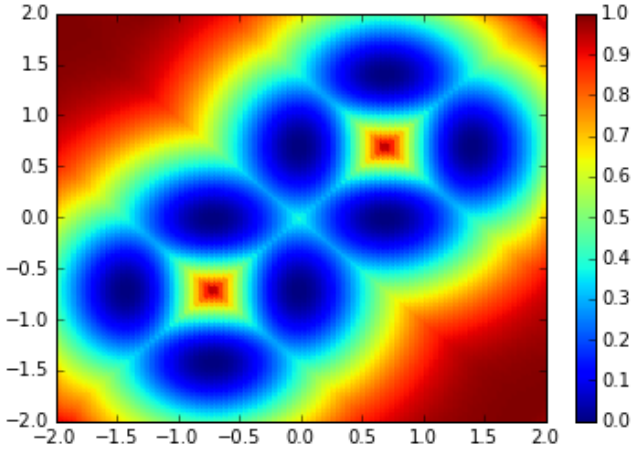


Figure 3.4: Using only the highest derivate.

the picture is clear. In the middle of both pictures a round place of low robustness can be seen though with the derivate approach it is significant smaller. The clear difference between both approaches is that in [2] everything appears to be less robust which might solely be because of their weighting system,especially the size of the area, but both approaches look similar.

From now on we will play with the definition of robustness and explore more possibilities with the derivate approach.Depending on the look on robustness one might say that a system is more robust if some small errors occur often, than if some large errors occur rarer. To make this more concrete, the current approach delivers the same robustness if there is a point with a derivate of 0.5 in one direction and 0 in the other direction and a point with a derivate of 0.25 in both directions. Though in infinite samples both points will show the same average deviation the first point is much more unstable, because a small change in the wrong direction will cause a worse result than a change in any direction at the other point. As there are several approaches to this problem the most simple one will be stated first and then more sophisticated ones. The most simple approach is to take only the highest absolute derivate as it shows the robustness in case of a worst case scenario. The result can be seen in Figure 3.4.

```
robustness = 0;
for outputDim = 1,2...,numOfOutputDims do
    for inputDim = 1,2...,numOfInputDims do
        derivate = abs(model.getDerivateValue(actualPoint,inputDim,outpuDim));
        if derivate > robustness then
            robustness = derivate
robustness = -robustness;
```

Algorithm 3: Taking only the highest derivate

Apparently the figure of this algorithms is not even close to Figure 3.3. This drastic change is due to points with derivatives being close to each other are seen as instable as a point with one extreme derivate and other much lower ones. Actually this worst case robustness is something which might be considered when looking into very brittle systems where things can break easily. But the focus here lies on a general robustness which requires some kind of trade off between worst case robustness and simply adding derivatives on top of each other. In the following algorithms will be presented which cover exactly this trade off. The most simple trade off can be achieved by simply adding the average and worst case on top of each other. To have some comparison a three pictures for trade off will be shown, which start at 0.25 and 0.75 and end at 0.75 and 0.25 as seen in Figure 3.5

The next approach is a simple mathematical trick to bring things which are close to each other even closer and especially favor the highest one. If one takes the square of a lot of values, adds them together and then takes the square root the result will be much closer to the highest value than by simply adding the normal values together. This can also be done

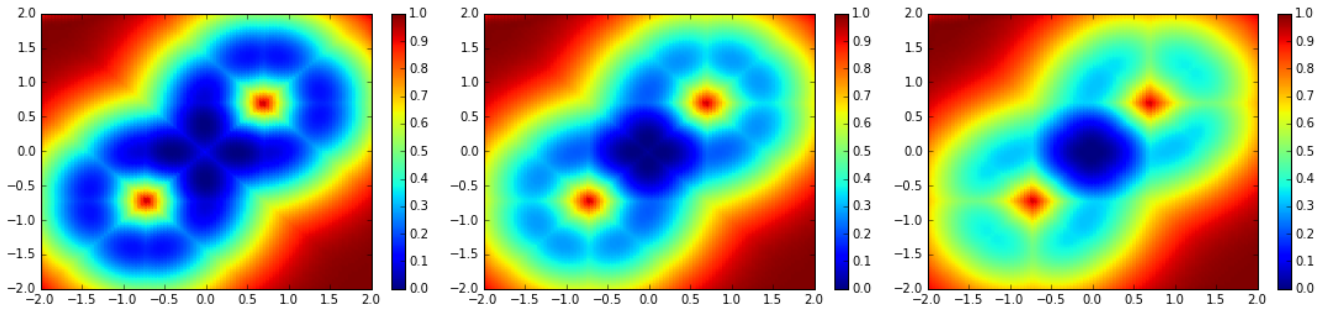


Figure 3.5: left to right: 0.25 sum - 0.75 highest, 0.5 sum - 0.5 highest, 0.75 sum - 0.25 highest

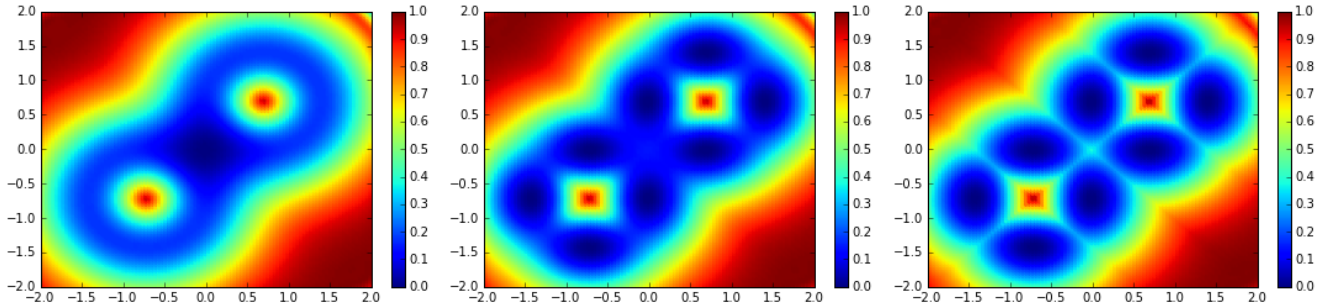


Figure 3.6: left to right: square, quartic, 64th order

with higher exponents than just the square. Next the algorithm for the square example will be pictured and the figures for square, quartic and 64th order will be shown to show the converging to highest value as seen in Figure 3.6.

```
robustness = 0;
for outputDim = 1,2...,numOfOutputDims do
    for inputDim = 1,2...,numOfInputDims do
        robustness = robustness + sqr(model.getDerivateValue(actualPoint,inputDim,outpuDim));
robustness = -sqrt(robustness);
```

Algorithm 4: Take the square adding up then square root

This was the exploration of the derivate approach. We looked into simple average, averaging the absolute values, taking a look at the worst case and finally trade offs between average and worst case.

3.2.2 Variance approach

In this approach the second possibility of robustness is described, namely robustness to changes to the value of the function without changes of the parameters $f(x) + \varepsilon(x)$. This robustness can be simply seen as the negative of the noise of the function, with some kind of scaling. Due to the changing nature of the noise, a heteroscedastic Gaussian Process is needed to correctly model the function. Only having the need to correctly calculate robustness, this approach assumes a fully trained heteroscedastic Gaussian Process model. In such a model, the robustness can simply be seen as the inverse of the variance, with some kind of scaling. This approach is very different from those before as here the model needs to be tuned in such a way that the function is fully known and no more uncertainties exist due to not having samples in a certain area. In such a case an aberration from the mean will be noticed as a variance and a higher variance will mean less robustness. If no such model exists the variance of the model can still be seen as an uncertainty of the model which either means, that the point is not robust, not learned or both. Such robustness can then be used to augment other approaches, such as those already presented. For visualization purposes we choose the MOP2 function from Equation (2.28), but we add noise in the form of $y1 = y1 + \mathbb{N}(0.01(x_0 + 2))$ and $y2 = y2 + \mathbb{N}(0.01(x_0 + 2))$. When calculating robustness by using the negative of the variance and combining it with other robustness approaches robustness is gained for a noisy function. If both approaches are weighted the same results for augmenting the approach of adding all absolute derivates up can be seen in Figure 3.7 and the augmented approach when using the square can be seen in Figure 3.8.

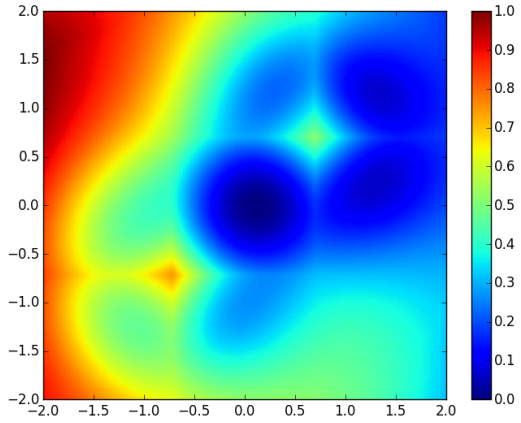


Figure 3.7: Absolute derivatives combined with negative variances

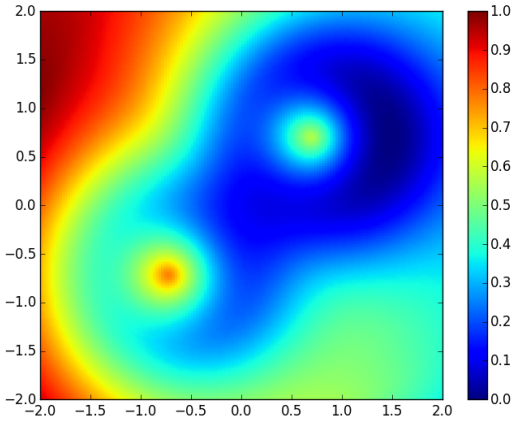


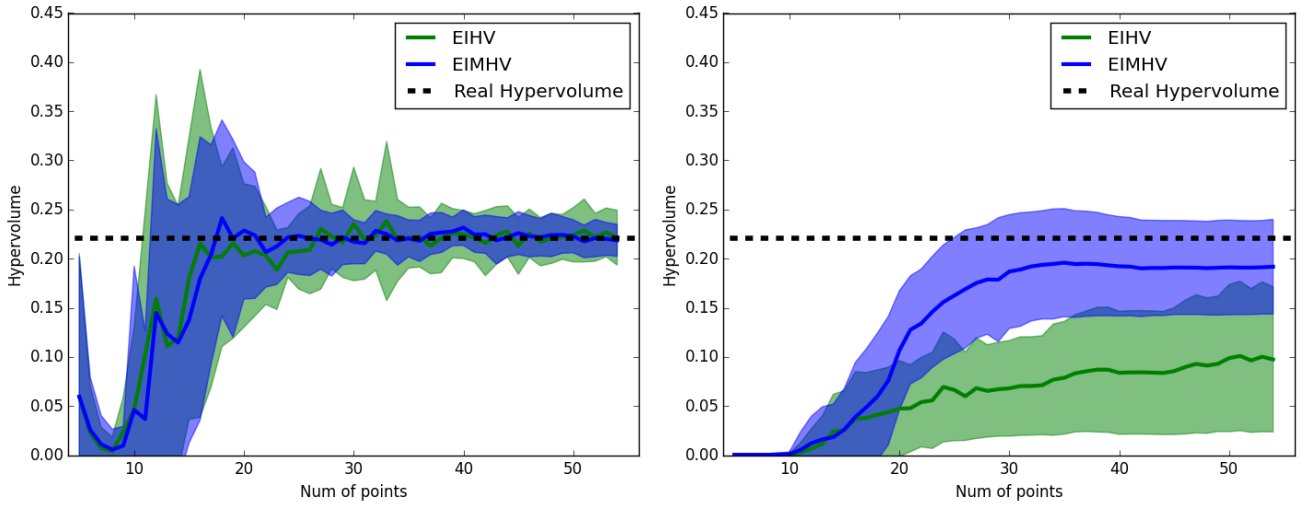
Figure 3.8: Square derivatives combined with negative variances

3.3 Improving EIHV

Improving expected improvement of hypervolume improves the quality of the chosen samples, which in turn lowers the amount of iterations from Bayesian optimization to gain the same result. The implementation of expected improvement of hypervolume [4] used in this thesis uses a Pareto set to split the space into volumes. In the paper [4] the Pareto set is gained from the observed values in objective space. The paper [2] on the other hand states, that the Pareto set gained from the model approximates the real Pareto front much better, than the Pareto set from observations. Calculating the expected improvement of hypervolume (EIHV) with the Pareto set gained from the observations will be called expected improvement of model hypervolume (EIMHV). Although [2] makes this statement for random evaluations and their shown results start with 20 random observation, it should be tested, if using a Pareto set from the model is better than using a Pareto set from observations in the EIHV implementation of [4].

3.3.1 Improving EIHV for a noiseless function

The most simple test is to test the difference in performance for the different Pareto sets while using a standard Gaussian Process to learn a function with no noise through Bayesian optimization. As for the noiseless function MOP2, described in Equation (2.28). Bayesian optimization was run by starting with 5 random points and then iterating 50 times. In each iteration the Pareto set of the model was calculated from 1000 random points evaluated by the model. For each, the model and the observations, the Pareto set was limited to 20 points to quicken the EIHV computation, although this only affects the Pareto set of the model due to the Pareto set of the observations not having more than 20 points. The Bayesian optimization described was run 2 times individually, but with the same start points for each using the Pareto set from observations and using the Pareto set from the model. To obtain statistically significant results both Bayesian optimizations were run 20 times.
Within the described Bayesian optimization, different measures can be made to see how well the function is learned. First of all the hypervolume of the Pareto set of the observations and the hypervolume of the Pareto set of the model were calculated as well as the root mean square error (RMSE). The RMSE was calculated both for 10000 random points and for the Pareto set of 10000 random points, as the direct target of Bayesian Optimization is only to learn the real Pareto front. Correctly learning the rest of the model is also important for secondary tasks like the robustness calculation described in Section 3.2. The resulting hypervolume of the Bayesian Optimization can be seen in Figure 3.9 along with its standard deviation.
When comparing both algorithms using the hypervolume bound by the learned model as measure of comparing as seen in Figure 3.9a, that the EIHV approach reaches the real hypervolume about two iterations earlier than the EIMHV approach. The EIMHV approach on the other hand is much more stable, which can be seen through the lower variance and the more exact fitting to the real hypervolume after about 20 iterations. In Figure 3.9b it can be seen, that the hypervolume of the observations is significantly higher when using EIMHV than when EIHV is used. A higher hypervolume in this case means, that more observations were made on or closer to the real Pareto front. Sampling on the real Pareto front should lead to a lower mistake when approximating only the real Pareto front, but may lead to a worse overall reproduction of the complete function. The steady rise in the hypervolume of the observed Pareto set when training through observation



(a) Hypervolume of the model

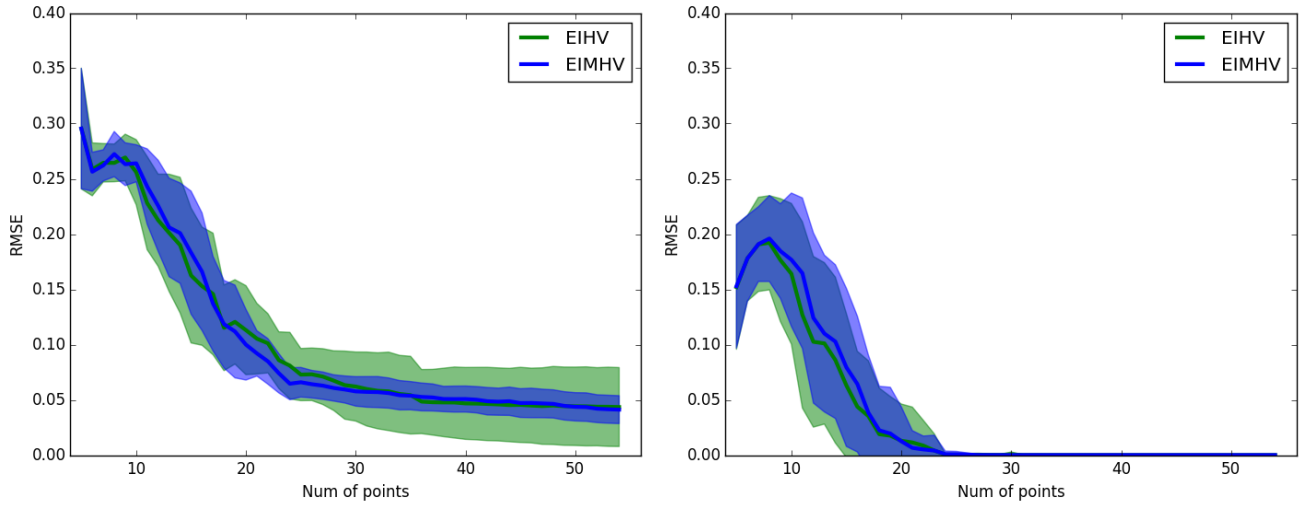
(b) Hypervolume of the observations

Figure 3.9: Hypervolume in Multi-objective Bayesian optimization for MOP2 with GP. The mean is shown as a line, while the standard deviation is plotted as a shaded area around the line.

shows, that sampling was not necessarily only done on the real Pareto front, but rather a little closer each iteration. To check the assumptions over made from observing the change in hypervolume the mean square error can be seen in Figure 3.10.
The assumption, that through learning with the Pareto set of the model, the function will be harder to model did not prove to be true. Rather the mean of both, learning with the observed or model Pareto set appear to be similar, while the variance of learning with the model Pareto set is better in longer runs seen in Figure 3.10a. Again the trend that in the beginning the observed Pareto set is slightly better can be seen in both Figure 3.10a, where the mean is better until about 18 points, and Figure 3.10b where it is better or equally good until 20 points. From these observation it can be derived, that first training with the observed Pareto set until about 15 to 20 points and then continuing with the model Pareto set may yield better results, than using only one method. To this point it appears, that the approach of [2] to use the model instead of the observations to calculate the Pareto set yields better results on the EIHV introduced by [4]. From looking directly into the results of each iteration, it can be seen, that the error from training with the Pareto set of the observations most likely yields from oversampling a certain area close to already made observations, which can be seen as a too low exploration within the algorithm. One iteration in which exactly this problem happened can be seen in Figure 3.11a.
In these figures all made observations are marked blue and the Pareto set points are red, while the point for the next observation is marked green, whereas the black points are 1000 random sampled points from the model, showing the models picture of the function. The line in the middle of the observations in Figure 3.11a existed since the 15th iteration and did not vanish due to only sampling close to where the green point is in the remaining 35 iterations. In Figure 3.11b on the other hand it can be seen, that the whole Pareto front is full of already observed points, validating the observation made for the hypervolume plots. So again using the model appears to be better than using only the observations.

3.3.2 Improving EIHV for a noisy function

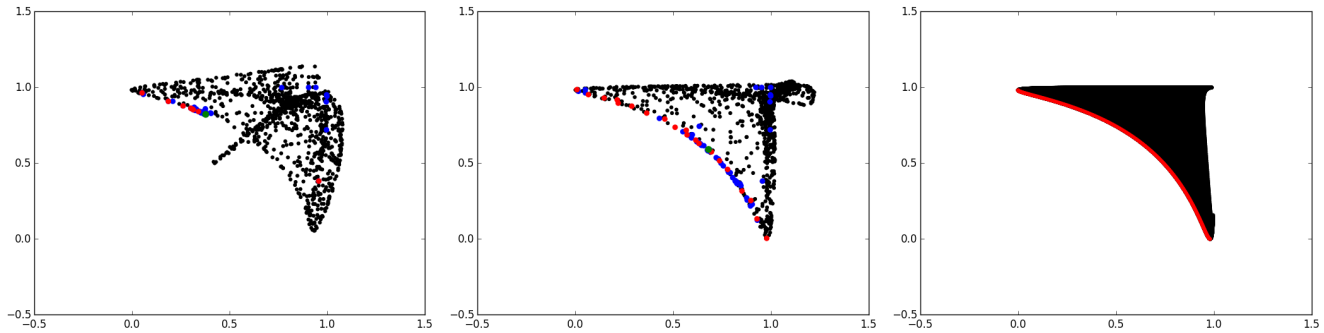
In [2] it is stated, that the modeled Pareto set is also better when the function which is to be modeled has noise. Again the MOP2 function from described in Equation (2.28) is used, but the noise terms $y1 = y1 + \mathbb{N}(0.01(x_0 + 2))$ and $y2 = y2 + \mathbb{N}(0.01(x_0 + 2))$ are introduced, creating an uneven noise over the Pareto front. The plots of the resulting hypervolumes can be seen in Figure 3.12.
Figure 3.12 shows, that Bayesian Optimization in general is a lot worse if there is noise on the model, since both methods yield worse results. It is to note, that both methods yield hypervolumes above the real hypervolume and are have much more variance, than their counterpart in Figure 3.9. The high variance can be explained through observation of the runs, while the wrong volume can be explained through adding of the noise. When observing the runs, training with the model was in overall more successful in the meaning, that only in two runs it was unable to create a model which produced similar visual results to the real function, while training with the observed Pareto set was leading to four runs without visual similarities in the result. These errors explain the high variance, since those visually wrong runs often produced a different hypervolume. If all runs with extreme errors were to be omitted, both training methods yield a much lower



(a) RMSE for 10000 samples of the real function

(b) RMSE for the Pareto Set of 10000 samples of the real function

Figure 3.10: Root Mean Square Error(RMSE) in Bayesian optimization for MOP2. The mean is shown as a line, while the standard deviation is plotted as a shaded area around the line.



(a) 50th iteration-BO with EIHV

(b) 50th iteration-BO with EIMHV

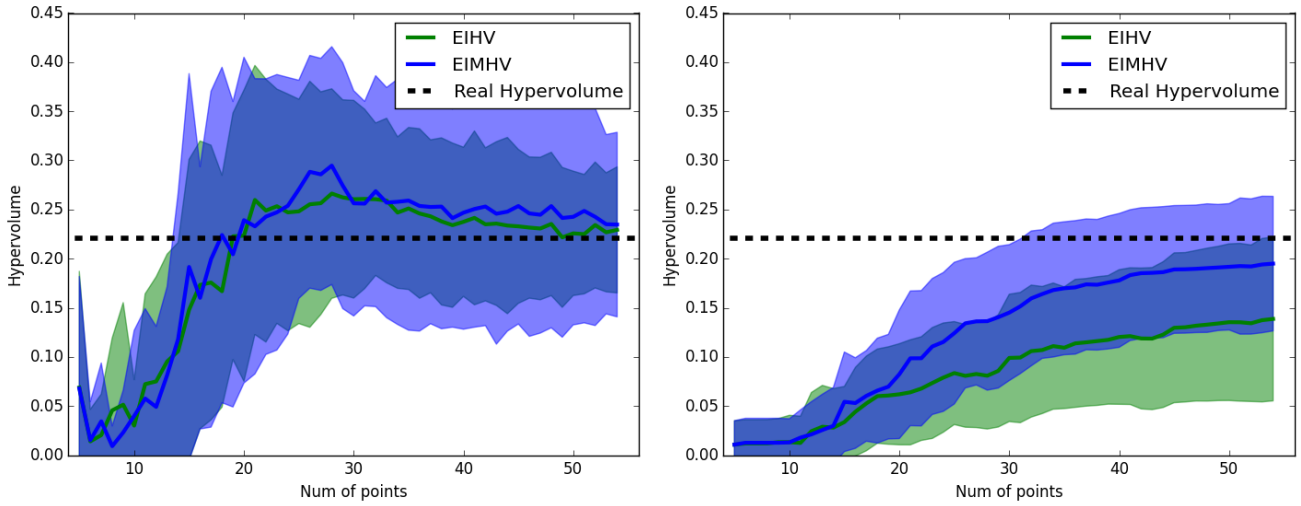
(c) MOP2 with its PF in red

Figure 3.11: Hypervolume in Bayesian optimization, random points from the model(black), Pareto set(red), observations(blue)

variance which is a lot closer to observations for the function without noise. The higher hypervolume can be explained due to the model believing, that the area in objective space is slightly bigger due to the noise. In this context it can be said, that training with expected improvement of model hypervolume is slightly more robust, but that in overall high noise with changes can not be modeled as good as without noise. As heteroscedastic Gaussian Processes are made for noise with changes the next comparison is between normal Gaussian Processes and heteroscedastic ones.
To calculate the mean and variance only ten runs were taken, since the calculation of the heteroscedastic Gaussian process is over twice as expensive as a standard Gaussian process. The easiest comparable results for those different processes are the root mean square error and the Negative Log Predictive Probability(NLPP) which is defined as

$$-logp(y = y^*|X, x^*, Y, \theta). \tag{3.1}$$

Whereas y^* equals the result from feeding x^* to the mop2 function with noise. X and Y are the data used to train the model, while θ are the parameters of the model. While the RMSE is a measure for the mean of the model, the NLPP measures both, mean and variance and is lower if both is closer to the real function. As training using the Pareto set of the model proved slightly more stable its RMSE from the new defined Bayesian optimization can be seen in Figure 3.13. It can be observed, that the RMSE is about the same in the end for the whole function, although the standard Gaussian Process proves to be a much better in the early runs and even slightly better in the later runs, while also having a lower variance. The RMSE of the standard Gaussian Process is overall better for the Pareto Front of the function. This difference can be explained, due to the heteroscedastic Gaussian process fitting poorer with a low amount of data points, which leads to a poorer approximation of the Pareto Front, which in turn leads to slower learning. Since the heteroscedastic Gaussian Process was worse for the RMSE, we take a look at the NLPP, which is a measure for how good the variance fits together with the mean in Figure 3.14.



(a) Hypervolume of the model

(b) Hypervolume of the observations

Figure 3.12: Hypervolume in Multi-objective Bayesian optimization for MOP2 with GP. The mean is shown as a line, while the standard deviation is plotted as a shaded area around the line.

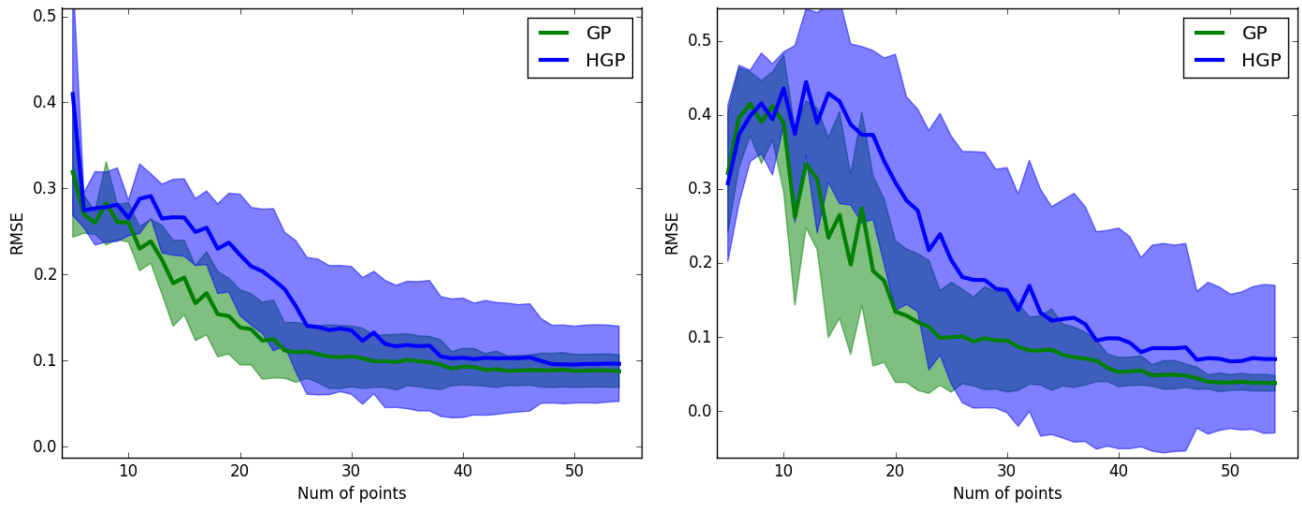
It can be seen, that as well for the whole function as for the Pareto front the heteroscedastic Gaussian process is better than the standard Gaussian process in regards of NLPP, while the difference in RMSE decreases with further iterations. The standard Gaussian Process performs better for the RMSE, while the heteroscedastic Gaussian process should be used to model the variance more exact. Furthermore, it shows that a changing variance can also be learned through this EIHV approach in general.
In internal runs using the Pareto set from the observation was tested as well, but performed worse so no new information was gained.

3.3.3 Results

Finally the conclusion which can be drawn from all observations over EIHV. The clearest result is that the Pareto set of the model should be used, when some iterations were made or some points already exist, the average amount of mistakes or in other words the variance is lower as seen in Figure 3.10 and Figure 3.11. In Figure 3.10 it can be seen, that this difference begins to show between 20 and 30 points or 15 and 25 iterations for a noiseless function. As already discussed, there appears not much of a difference in performance between both Pareto sets when going into functions with a lot of noise, which, with the observation of when the difference shows for a noiseless function, leads to the conclusion that at least after 30 points or 25 iterations the Pareto set of the model should be used.
A less extreme difference of performance can be seen in Figure 3.9 and Figure 3.10 where learning with the Pareto set of the observation perfomrs slightly better in early runs, which is why we suggest to first use the observed Pareto set until at most 25 iterations and then switch to the Pareto set of the model. If only one type can be chosen for some reason and over 25 iterations are to be made training with the Pareto set of the model is always recommended.
Additionally it was shown, that the heteroscedastic Gaussian Process from [7] works with the EIHV approach from [4] for functions with a lot of noise, as well as performing better than the standard Gaussian Process in terms of modeling the variance. Since the standard Gaussian Process performs better in the EIHV for functions with a lot of noise in the early iterations and is faster it is recommended to first use the standard Gaussian Process and then switch to the hereroscedastic Gaussian process.

3.4 Gaining robustness through HGP in MOO

In Section 3.3 it was shown that the heteroscedastic Gaussian process model is able to run alongside the EIHV approach from [4]. While in Section 3.2.2 the general idea of how gaining robustness together with a heteroscedastic Gaussian Process works in general is pictured, this section focuses on the advantages over a standard Gaussian Process in multi-objective optimization. The value of the function is considered as one objective, while the other objective is the robustness itself. In this context it is to be expected, that the heteroscedastic Gaussian process adds much more information. To evaluate the difference in performance, each model learns a function $y = sin(4\pi x) - x$ to which the noise $\epsilon = \mathcal{N}(0, 2x)$

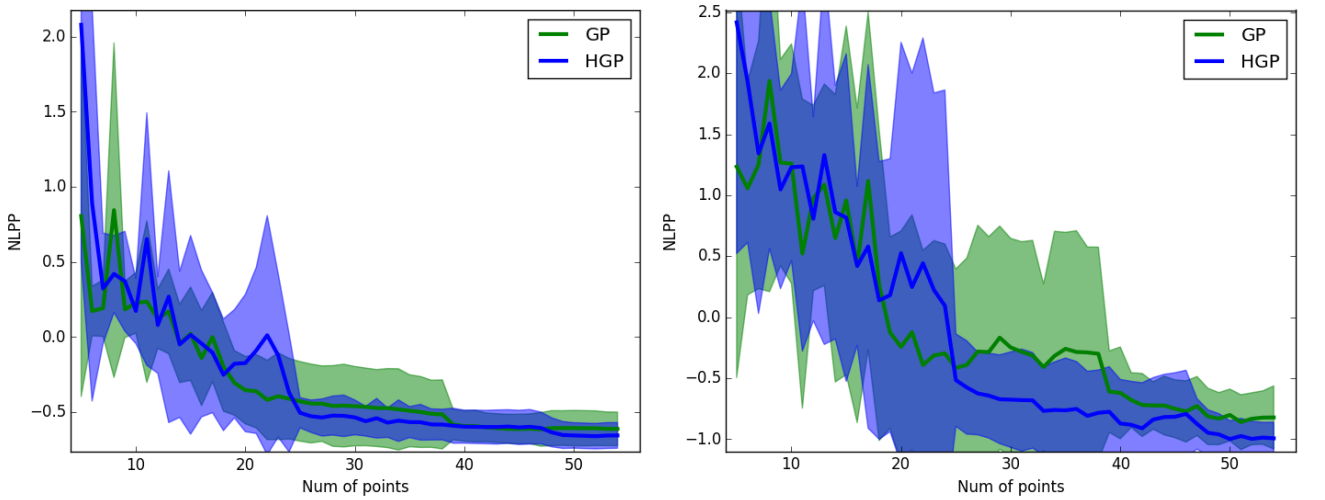


(a) RMSE for the whole function

(b) RMSE for the Pareto front of the function

Figure 3.13: Root mean square error(RMSE) in Bayesian optimization(GP vs HGP) for MOP2 with noise. The mean is shown as a line, while the standard deviation is plotted as a shaded area around the line.

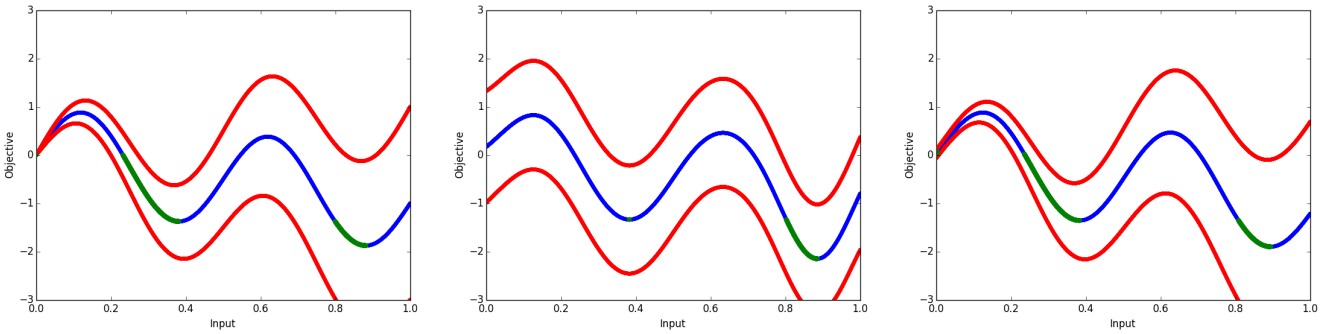
is added. The data derived from the 300 random samples will then be compared to the real function. Experimental results for the parameter space can be seen in Figure 3.15 and in Figure 3.16 for the objective space.
It can be observed, that the standard Gaussian process model is assuming constant variance and only shows a Pareto front due to low uncertainty and thus can not make any predictions about the real robustness. The heteroscedastic Gaussian process shows almost no difference to the variance of the real function and therefore is able to use robustness as an additional objective in multi-objective optimization. The observed results proof the usefulness of heteroscedastic Gaussian processes when modeling robustness in multi-objective optimization.



(a) NLPP for the whole function

(b) NLPP for the Pareto front of the function

Figure 3.14: Negative Log Predictive Probability (NLPP) in Bayesian optimization (GP vs HGP) for MOP2 with noise. The mean is shown as a line, while the standard deviation is plotted as a shaded area around the line.

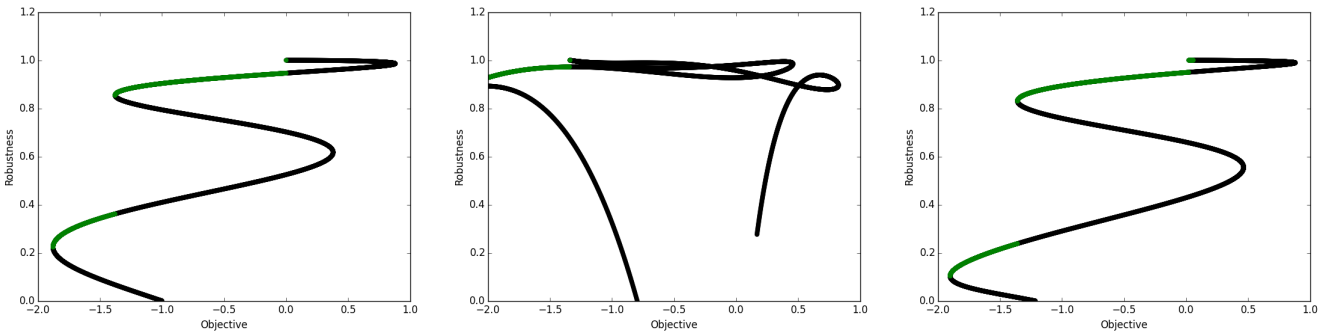


(a) Real function

(b) Gaussian process

(c) Heteroscedastic Gaussian process

Figure 3.15: 1D function with linear noise in parameter space. The mean of the function is displayed in blue, while the standard deviation is shown in red and the Pareto front is colored green.



(a) Real function

(b) Gaussian process

(c) Heteroscedastic Gaussian process

Figure 3.16: 1D function with linear noise in objective space with robustness as objective. The whole function is shown in black, while the Pareto front is green

4 Conclusion

The goal of this thesis was to model robustness in multi-objective optimization, especially for robotic systems and enable robustness as an additional objective. Robustness is needed to know which parameter configurations have predictable behavior and for those which don't it is a measure for the uncertainty in predicting and models are used to calculate the robustness. As it is sometimes impossible to acquire accurate models for robots analytically, machine learning models were introduced to enable passive learning of all, for the specified task, relevant properties which can be derived from the given parameters. These machine learning models are Gaussian processes as well as heteroscedastic Gaussian processes. Since sampling robots is expensive Bayesian optimization was added to the algorithms, as Bayesian optimization minimizes the amount of samples needed to learn an accurate model. To accurately learn and describe multiple objectives, multi-objective optimization and the definition of the Pareto front were included. Furthermore Bayesian optimization was extended to deal with multiple objectives through the use of the expected improvement of hypervolume [4].

All methods and algorithms mentioned in this thesis were implemented into a framework, developed in the programming language Python. In Section 3.2 several different approaches for calculating robustness were presented and discussed. It was shown,that the algorithm for computing robustness depends on the task, while all presented approaches based on derivatives only show robustness correct within a defined unknown area due to essentially using Taylor approximation. In Section 3.3 methods for improving EIHV were discussed. By averaging over 20 Bayesian optimization runs using EIHV we validated, that better results can be achieved after 25 iterations, by using the Pareto set of the model instead of the Pareto set of the observations. Data from the same runs suggested, that using the Pareto set of the observations before 15 iterations yields better results, than using the Pareto set of the model. When comparing Gaussian processes to heteroscedastic Gaussian processes the data also suggested, that using a standard Gaussian process for the first few iterations performs better than a heteroscedastic Gaussian process, but after these iterations the heteroscedastic Gaussian process has the advantage of modeling the noise better.

Through the usage of Gaussian Process models in Bayesian optimization, multi-objective models can be created with low amount of observations and then used to calculate robustness. Additionally, multi-objective Bayesian optimization was improved by introducing a novel acquisition function (EIMHV) improving upon the EIHV from [4].

4.1 Future Work

Although several computations for robustness were presented, not all aspects of robustness were observed. In the approaches using derivatives it remains to be tested how the adding of higher derivatives influences the computation of the robustness. In this context it would be of interest to analyze up to which order derivatives are needed to deliver a similar result as [2] and if it is even possible to deliver the same or a very similar result only with derivatives, as they are not including noise on the system. A possibility for calculating those higher order derivatives would be to calculate them from samples of the model. Additionally to using derivatives and the variance also other approaches which sample defined areas as [2] remain to be observed. When calculating robustness a scale to measure the robustness, as degrees exist for temperatures, was left to be desired. Instead it is common practice to define the lowest robustness as zero and the highest as one. This may lead to the illusion, that two systems, whereas one is extremely stable and the other is extremely unstable have a similar robustness.
While improving the EIHV approach from [4] it was observed, that using the Pareto set of the model is more stable after a certain amount of iterations. This observation was validated through showing a clear difference when averaging over 20 observations. Other observations like the Pareto set of the observations performing better before a certain amount of iterations, cannot be seen as validated due to only a small difference in performance while averaging over ten observations. These other observations need to be validated on a bigger data set. The not validated observations include, that the Pareto set of the observation performs better before 15-25 iterations, that the mean of the standard Gaussian process is better for functions with noise with less than 15 iterations, than the mean of the heteroscedastic Gaussian process and that the heteroscedastic Gaussian process performs better with noise functions in Bayesian optimization after a lot of iterations. Finally some conclusions which need to be validated, with the first being, that using the Pareto set from observation in early iterations, then switching to the Pareto set of the model in EIHV leads to improved results. And at last the conclusion that first training with a Gaussian Process and then a heteroscedastic Gaussian process leads to an improved result needs to be validated.

Bibliography

[1] D. A. Van Veldhuizen and G. B. Lamont, "Multiobjective evolutionary algorithm test suites," in *Proceedings of the 1999 ACM symposium on Applied computing*, pp. 351–357, ACM, 1999.

[2] R. Calandra, J. Peters, and M. Deisenrothy, "Pareto front modeling for sensitivity analysis in multi-objective bayesian optimization," in *NIPS Workshop on Bayesian Optimization*, vol. 5, 2014.

[3] M. Tesch, J. Schneider, and H. Choset, "Expensive multiobjective optimization for robotics," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 973–980, IEEE, 2013.

[4] M. Emmerich and J. Klinkenberg, "The computation of the expected improvement in dominated hypervolume of pareto front approximations," *Rapport technique, Leiden University*, 2008.

[5] "Gpy." `http://sheffieldml.github.io/GPy/`. Accessed: 2016-03-14.

[6] P. W. Goldberg, C. K. Williams, and C. M. Bishop, "Regression with input-dependent noise: A gaussian process treatment," *Advances in neural information processing systems*, vol. 10, pp. 493–499, 1997.

[7] M. K. Titsias and M. Lázaro-gredilla, "Variational heteroscedastic gaussian process regression," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 841–848, 2011.

[8] E. Kim, "Everything you wanted to know about the kernel trick (but were too afraid to ask)." `http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html`, September 2013. Accessed: 2016-01-25.

[9] C. E. Rasmussen and C. K. I. Williams, "Gaussian processes for machine learning," 2006.

[10] Q. V. Le, A. J. Smola, and S. Canu, "Heteroscedastic gaussian process regression," in *Proceedings of the 22nd international conference on Machine learning*, pp. 489–496, ACM, 2005.