Learning Generalizable Models for Compliant Robots

Lernen generalisierbarer Modelle für nachgiebige Roboter

Bachelor-Thesis von Mike Matthias Smyk aus Offenbach am Main

Oktober 2014

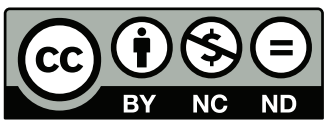
TECHNISCHE
UNIVERSITÄT
DARMSTADT

Learning Generalizable Models for Compliant Robots
Lernen generalisierbarer Modelle für nachgiebige Roboter

Vorgelegte Bachelor-Thesis von Mike Matthias Smyk aus Offenbach am Main

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Herke van Hoof
3. Gutachten:

Tag der Einreichung:



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 10. Oktober 2014

(Mike Matthias Smyk)

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, October 10, 2014

(Mike Matthias Smyk)

Acknowledgments

I would like to thank my supervisor Herke van Hoof for the excellent mentoring and for the plenty of his time he spend for our regular meetings, as for the time when I needed his advice additionally.

I would like to thank all the guys from "the office", who cheered the time spending on working on the thesis a little bit up. Especially I want to thank Christos Votzkos for his advices and his help when I got stuck with problems.

Last but not least I want to thank my friends and my family for all the support they gave and give me during the whole studies.
Danke für die Unterstützung die ihr mir tagtäglich gebt, sei es im Studium oder in allen anderen Lebenssituationen.

Abstract

Generating models for robots is a typical problem in robotics. The behavior of a compliant robot changes when it lifts an object. So for every object the robot should lift a separate model is needed. To tackle this problem multi-task Gaussian Processes (multi-task GP) can be used to learn one generalizable model, by observing the data for a set of objects (respectively tasks). To adapt the model for a new unseen object, only a few data points have to be gathered to make good estimates.

In this thesis multi-task GPs are introduced and applied to data of a real compliant robot (the Robotino XT) to learn a dynamics model (forward and inverse). The multi-task GPs are evaluated under several settings. Firstly, two objectives for the optimization of the hyper-parameters are analyzed: maximizing the Log-Likelihood of the training data and cross-validation on this data. Secondly, two different approaches to model the task-similarities are compared: directly optimizing the similarities of the task and defining feature representations for the tasks. Furthermore, the influence of different amount of training points for a new task, as different number of known objects are investigated.

It shows that if there is a high amount of data for several tasks and just a few data points for a new task, multi-task GPs yield improved results compared to single GP approaches.

Zusammenfassung

Das Erstellen von Modellen ist ein typisches Problem in der Robotik. Allerdings verändert sich das Verhalten von nachgiebiegen Robotern, wenn sie ein Objekt anheben. Daher müsste man für jedes Objekt ein separates Modell erstellen. Um dies zu verhindern können multi-task Gauss Prozesse (multi-task GP) verwendet werden um ein generalisierbares Modell zu lernen. Dabei werden für eine Menge von Objekten Daten gesammelt. Soll der Roboter nun ein neues ungesehenes Objekt anheben, müssen dafür nur ein paar wenige Daten gesammelt werden um gute Vorhersagen machen zu können.

In dieser Arbeit werden multi-task GPs eingeführt und auf Daten eines realen nachgiebigen Roboter (den Robotino XT) angewendet, um für diesen ein dynamik Modell zu lernen (vorwärts, sowie invers). Die multi-task GPs werden dabei unter diversen Szernarien evaluiert. Es werden zwei Zielfunktionen für die Optimierung der Hyper-Parameter analysiert, zum einen das Maximieren der Logarithmischen Wahrscheinlichkeit auf den trainings Daten und zum anderen die Kreuzvalidierung auf diesen Daten. Zwei Ansätze die task-similarities zu modellieren werden veerglichen, zum einen werden die Ähnnlichkeiten direkt optimiert und zum anderen werden die tasks in den Feature-Raum projeziert um ihre Ähnlichkeiten dort zu berechnen. Außerdem werden der Einfluss von verschiedenen Mengen an Datenpunkten, die für ein neues Objekt zur Verfügung stehen, sowie verschiedene Anzahlen von vorher gesehenen Objekten untersucht.

Es zeigt sich, dass für Fälle in denen für das neue Objekt nur sehr wenige Datenpunkte vorhanden sind, man die Ergebnisse der Modelle mit multi-task GPs im Vergleich zu einzel GP Ansätzen verbessern kann.

Contents

List of Figures

1 Introduction

Many strategies for controlling a robot require a detailed model of the robot. Suppose a robot should reach a certain state given a database of states that is generated by observing the robots behavior. The robot can not reach this state if it is not in the database, since there is no information about how to reach it. So there is a need for generating models, since it is not possible to cover all potential states of a robot.

In robotics there are two main types of models, the kinematics and the dynamics model, whereas for each of the two there is a forward and a inverse case. The forward kinematics model is a function that calculates the position of the robots end-effector by its joint angles. The forward dynamics model is a function that calculates the joint angles or the next state of a robot by torques that are applied to the robot. The inverse cases are defined accordingly. Inverse kinematics computes the joint angles that are needed to reach a certain position, whereas inverse dynamics computes the torques that have to be applied to reach certain joint angles, velocities and accelerations.

Generating such models for robots is a typical problem in robotics. Classical industrial robots, for example, are designed to have analytically solvable kinematics and dynamics. Their six Degrees of Freedom (DoF) are ordered in a way that singularities are avoided and there is a solution for the inverse kinematics and dynamics everywhere in the reachable space (workspace).

Another approach to get models for robots is to generate them geometrically [1, p.97]. An easy example for this is the 2-DoF Scara Manipulator [1, p.15] for which a forward kinematics model can be generated just by investigating the angles of the joints in a geometrical way.

For more complex robots these two approaches are not feasible, since for example a consolidated knowledge of the physical influences has to be known. Instead, the behavior can be learned by machine learning techniques such as Reinforcement Learning (RL) [2]. In RL behavior is learned by applying multiple actions to the robot that should reach a certain state and rewarding "good" actions. With this method appropriate controllers can be learned, but the actions have to be applied to the real robot which is very time consuming and wears the robot out. Finding a suitable reward function, i.e. defining what is a "good" action, is also an occurring problem. Instead, there are also model learning techniques that learn the models on the data, like Gaussian Processes (GPs). There the model learning problem is reduced to a regression problem.

Some models are only valid in certain contexts. An example for a model that can be learned by GPs is driving a car with a certain velocity $\dot{v}_c$. Under the assumption that a model should be learned which accelerates the car so that its velocity in the next time step is $\dot{v}_c$ and the gradient of the street is constant. If all the data is gathered driving on a street with zero gradient, this model is unsuitable for a car driving on a street with a different gradient. As soon as the gradient of the street changes the model has to be re-learned, which implies gathering more data for the new environment. Then

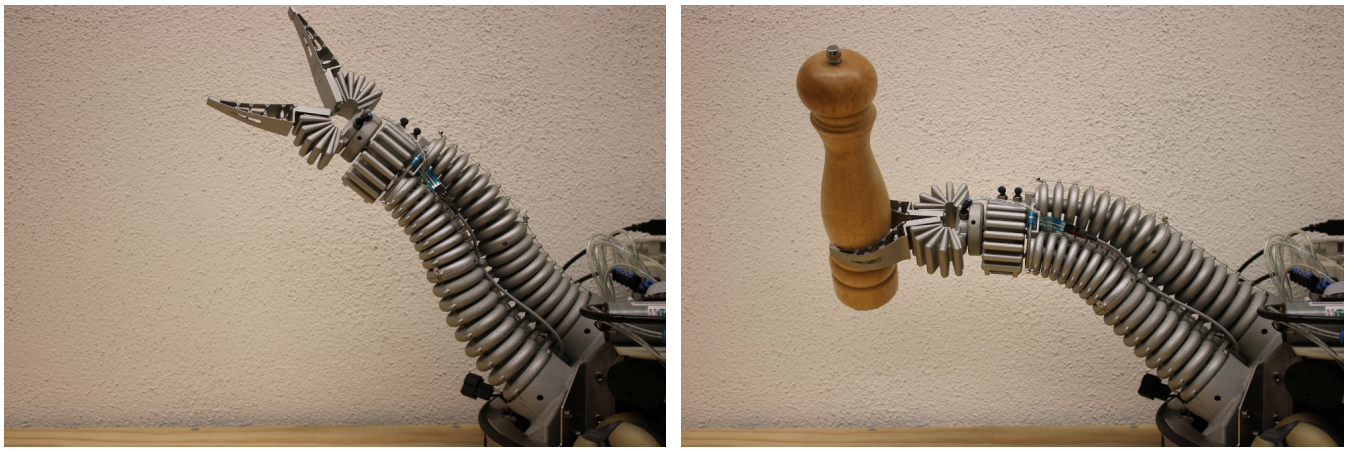


Figure 1.1: Compliant robot without and with load gripped. In both pictures the same control input is applied

there are two models for each of the gradients. Any new gradient has to be re-learned once again. It is not possible and not desirable to learn a model for every thinkable gradient. A model that generalizes over environments and adapts just by seeing few new data points would be needed.

There is a similar problem with compliant robots. Since they are not rigid, as soon as there is an external force applied, the robot bends, i.e. its dynamics changes, and a learned model has to be re-learned. A possible external force can be applied by lifted objects. Figure 1.1 shows an example of a compliant robot lifting an object. In both images the same control inputs are applied to the robot, but the position of the end-effector is different.

The advantage of compliant robots is that they are lightweights and can be used in presence of humans, whereas rigid robots can just be used with a certain safety distance. If a compliant robot hits a person it will yield in contrast to rigid robots. On the other hand this is also the problem trying to learn dynamics models for such robots. Lifting objects is a common task for robots, but as Figure 1.1 illustrates, the object strongly affects the dynamics. As in the car example it is not possible to learn a model for each thinkable object to be lifted.

An example for a compliant robot is the Robotino XT. It is a pneumatically actuated robot, that has a trunk, called a Bionic Handling Assistant (BHA), on a drivable platform. The BHA has six chambers (actuators) that can individually be set on different pressures up to 1.5 bar. It has a gripper with which it can grab different loads up to 600 grams[1] and six cable potentiometers with a cable to the end of each chamber to measure the actuator lengths of the BHA (see Figure 1.2).

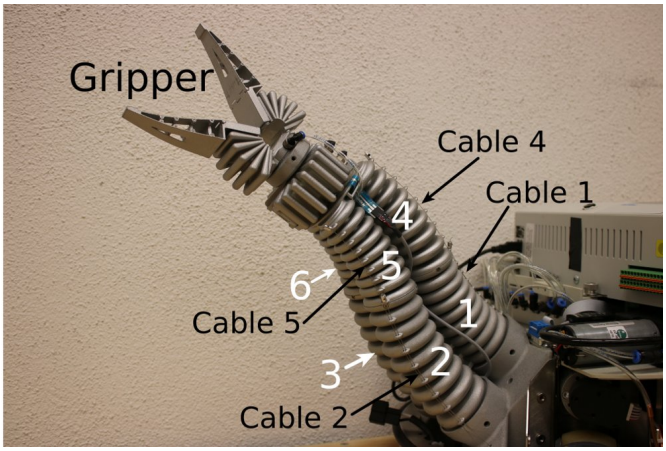


Figure 1.2: Robotino XT - Bionic Handling Assistant (BHA). The white numbers are denoting the numeration of the actuators

In case of this pneumatically driven robot, a forward kinematics model computes the position of the end-effector (the gripper) given the actuator lengths (cable lengths) of the BHA. Accordingly, the inverse kinematics model computes the actuator lengths that are required to reach a certain position. In comparison to the classical notion of dynamics, where the state in the next time step is estimated given a control input that is applied to the robot (forward model), here not the next time step but the resulting actuator lengths given a control input to hold the state is estimated. Analogically in the inverse model the control input is estimated to obtain a certain robot state (actuator lengths), not in the next time step, but at all.

The forward kinematics for the BHA can be modeled by using geometric approaches, as in [3] or [4]. Even if the robot lifts an object this model does not change, since the lengths of the cables still indicate the position of the end-effector. So there is no need for a generalizable model.

Bischoff et al. [5] learn a forward dynamics model for the Robotino, whereas they investigate an object pick-up task. Their challenge is to drive to an object and pick it up. However, this work is about lifting the picked up object to a certain position.

In [6] Rolf and Steil develop an inverse kinematics model for the BHA and the "first functional control concept for this challenging platform", whereas they use an extended version of the BHA with 3 sections (the BHA in Figure 1.2 has only

[1] Robotino XT Manual, Festo AG & Co. KG

2 parts). They only learn a model for the empty trunk.

So there are methods to learn models for the BHA with an empty trunk. As mentioned, if the BHA lifts an object the model changes and the dynamics has to be re-learned, i.e. data has to be gathered once more, which is very time consuming. Assuming that there are already learned models for lifting some objects, it should be possible to use these to make predictions for a new, never seen object, by just gathering some few data points.

Adapting the model to a new object requires generalization over models. Such a model can be learned by using multi-task Gaussian Processes (multi-task GPs). In the following lifting a specific object is called a task. Making predictions using multi-task GPs is described in [7]. In comparison to standard Gaussian Processes, for each estimate not just the data of one task is considered, but also the data of other related tasks weighted by the corresponding task-similarity.

To model these task-similarities different approaches can be used. All the approaches generate a task-similarity matrix. In [7] and [8] the values of the task-similarity matrix are directly set. To optimize the matrix a EM-Algorithm is used. The drawback of this approach is, it has to be ensured that the resulting matrix is positive definite, since it has to be a covariance matrix. Whereas in [9] and [10] feature representations of the tasks are defined. The advantage is that the similarities can be calculated in feature space with a similarity function. In [9] this features are defined manually and in [10] the multi-task GPs are used for a different application, where not the relation between tasks is investigated but it is tried to model multiple related functions in one. In this work the estimates for one task should be improved by using the information from other known tasks.

In [8] Chai et al. learn an inverse dynamics model for a rigid 6 DoF robot. They used to lift different objects of different weights and generalize over the resulting models using multi-task GPs. The advantage in using a rigid robot is, that lifting an object only changes the inertia parameters of the last link. Which is an assumption that does not hold using compliant robots, where the dynamics of the whole robot changes.

The idea is to learn models for several tasks. For a new task, just a few data points has to be gathered to estimate the task-similarity and adapt the model. This somehow is inspired by the way humans lift objects [11]. If a human being has no or wrong prior knowledge of an objects weight, since it is heavier than it looks, he first applies some forces to his arms trying to lift the object. One can interpret this as the human is collecting data for the new object and then adapts his internal model to the weight of the object to lift it.

2 Multi-Task Gaussian Processes

Gaussian Processes (GP) are used to solve regression problems. The benefit of GPs is that for each estimated value there is an approximation of the uncertainty. In the following chapter Gaussian Processes, the special case of multi-task Gaussian Processes and hyper-parameter optimization techniques are introduced.

2.1 Gaussian Processes

Gaussian Processes (GP) are an appropriate method to solve regression problems. For small and intermediate sized data sets GPs are able to perform good estimates [12]. For huge data sets they are computationally inefficient.

GPs can be seen as distributions over functions. They describe how likely certain function values at certain points are. In areas with many training points it is more likely to make a correct estimate of the function value than in areas without training points.

Let $\mathbf{K}(\mathbf{A}, \mathbf{B})$ be the Gram matrix, i.e. each entry is the similarity of each vector $\mathbf{a}$ of matrix $\mathbf{A}$ to each vector $\mathbf{b}$ of matrix $\mathbf{B}$, so entry $\mathbf{K}_{ij}$ is computed by $k(\mathbf{a}_i, \mathbf{b}_j)$, whereas $\mathbf{a}_i$ and $\mathbf{b}_j$ are vectors and k is called a covariance function. Further let $\mathbf{X}_* = [\mathbf{x}_{*1}, \mathbf{x}_{*2}, \ldots, \mathbf{x}_{*L}]$ be a matrix of L test inputs and $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$ a matrix of N training inputs. Before any training points are seen ($N = 0$), the GP with zero mean $\mathcal{GP}(0, \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*))$ is called the GP Prior, which is the prior knowledge that is available for the test inputs.

The covariance function k, also called a kernel function, calculates the similarity of two input values. Given two inputs $\mathbf{x}_p, \mathbf{x}_q$ the covariance function $k(\mathbf{x}_p, \mathbf{x}_q)$ denotes how related the inputs are to each other. One covariance function is the Squared Exponential Kernel, which provides a measure for the distance of two inputs:

$$k(\mathbf{x}_p, \mathbf{x}_q) = \sigma_f^2 \exp\left\{-\frac{||\mathbf{x}_p - \mathbf{x}_q||_2^2}{2l^2}\right\} \tag{2.1}$$

Where σ_f and l are called hyper-parameters of the kernel. σ_f denotes the variance of the kernel function, i.e. for a value greater than one the Gaussian is stretched and with a value less then one the Gaussian is shrunk. l is the length scale, which influences the range in which data points are considered to be similar.

A more detailed introduction and explanation of the influence of the parameters, as other examples for kernel functions can be found in [13] or [14].

Figure 2.1 shows the GP prior estimates for some sample points $\mathbf{X}_*$. The thick black line denotes the mean function, which is the mean estimate the GP makes. The gray area is two time the standard deviation of the estimate, this denotes the certainty of the GP making a guess (the mean). And the thin colored lines are some randomly sampled functions. As mentioned, GPs are distributions over functions and the colored functions are some samples of this distribution.

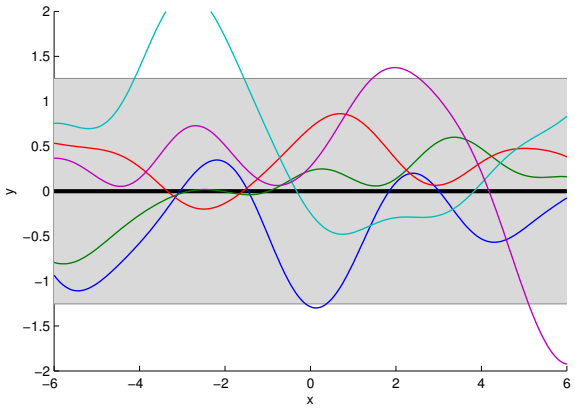


Figure 2.1: GP Prior. The thick black line is the mean function, the shaded area is two time the standard deviation of the distribution and the small colored lines are sample functions of the distribution

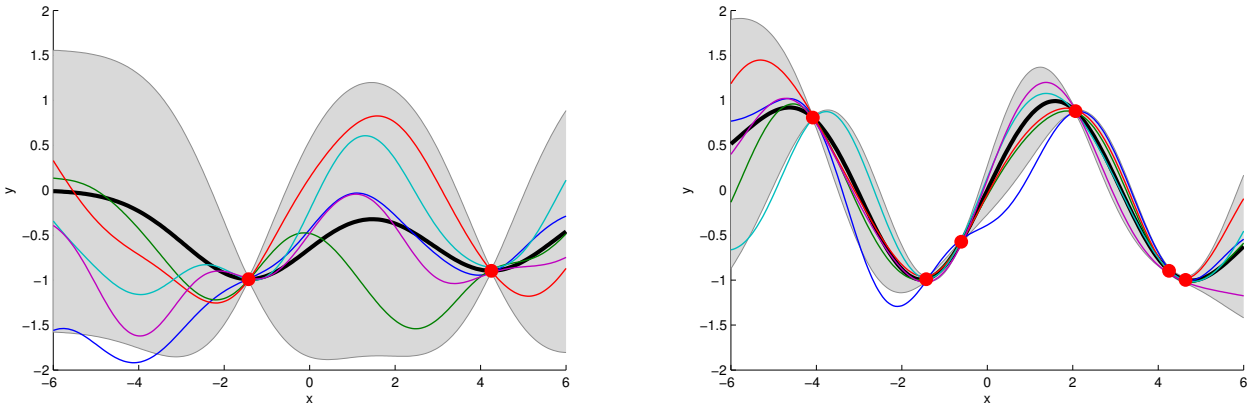
This is the prior knowledge a GP has. Adding training data to the GP conditions the distribution, so only functions that go through this via points are part of the distribution. The mean of the noiseless conditioned GP is then

$$\bar{\mathbf{f}}_* = \mathbf{K}(\mathbf{X}_*, \mathbf{X})\mathbf{K}(\mathbf{X}, \mathbf{X})^{-1}\mathbf{y} \tag{2.2}$$

and the covariance

$$\text{cov}(\mathbf{f}_*) = \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*) - \mathbf{K}(\mathbf{X}_*, \mathbf{X})\mathbf{K}(\mathbf{X}, \mathbf{X})^{-1}\mathbf{K}(\mathbf{X}, \mathbf{X}_*) \tag{2.3}$$

So for all test points the similarity to the training points is computed. The more similar they are the stronger they are correlated to the function value of this training points. The similarity is again computed by the covariance function. Figure 2.2 shows a GP conditioned on two and six training points.



(a) Two training points

(b) Six training points

Figure 2.2: GP posterior. The thick black line is the mean function, the red dots are training points sampled from an underlying function, the shaded area is two time the standard deviation of the distribution and the small colored lines are sample functions of the distribution

The estimates shown in Figure 2.2 assume that there is no noise on the data. Since all the training data is sampled from an underlying function, this assumption is suitable. For data gathered in the real world, it always has to be assumed that there is noise on the data, because of noise of the sensors for example. To model this noise one more parameters is introduced: σ_n, which is an estimate of the expected noise on the data. It is added to the diagonal of the Gram matrix. The equation for the mean and the covariance changes as follows [13, p.16]:

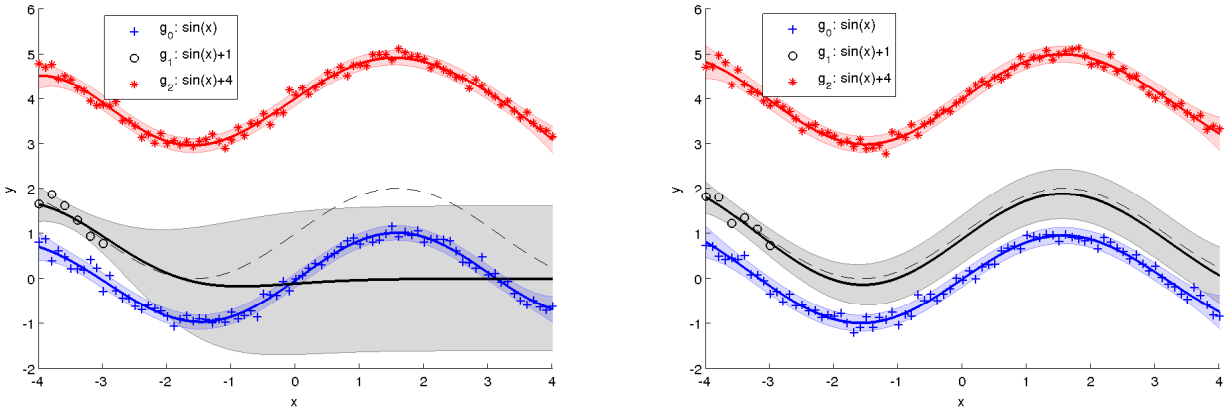
$$\bar{\mathbf{f}}_* = \mathbf{K}(\mathbf{X}_*, \mathbf{X})\left[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n\mathbf{I}\right]^{-1}\mathbf{y} \tag{2.4}$$

$$\text{cov}(\mathbf{f}_*) = \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*) - \mathbf{K}(\mathbf{X}_*, \mathbf{X})\left[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n\mathbf{I}\right]^{-1}\mathbf{K}(\mathbf{X}, \mathbf{X}_*) \tag{2.5}$$

Where $\mathbf{I}$ is the identity matrix with same size as $\mathbf{K}(\mathbf{X}, \mathbf{X})$.

2.2 Multi-Task Gaussian Processes

A specialization of Gaussian processes are multi-task Gaussian Processes. They allow generalizing over several functions, respectively tasks, so that the estimates for functions with fewer data get better by also considering the training points of the other functions.



(a) Three single GPs **(b)** one multi-task GP

Figure 2.3: Multi-task GP example. The thick lines are the mean predictions for the corresponding tasks, the dashed line is the underlying function of task 1 that should be predicted and the shaded areas are two times the standard deviation

Figure 2.3 shows an example of a regression problem. In this example are three underlying functions that should be estimated: $g_0 = \sin(x)$, $g_1 = \sin(x) + 1$ and $g_2 = \sin(x) + 4$. In the following the functions are called tasks. To simulate training data, some values are sampled and their function value is calculated adding some noise. For g_1 and g_3 there is much data generated for training, for g_1 there are only 6 data points.

Standard single GPs for each task compute estimates as shown in Figure 2.3a. For the two tasks with big amount of training data the GPs adapt very well to the training data and so to the underlying function. In comparison g_1 has only limited information (training data) so the estimates in the area without training data are far away from the real values.

With the assumption that the shape of g_1 is similar to the other tasks with many training points, the results for the range without training data can be improved. The outputs of the multi-task GPs are not only conditioned on the training points of one task but also by the outputs of the other tasks. The outputs of the other task are weighted by so called task-similarities. Task-similarities are a measure of how similar one function (or task) is to another function (or task). The effect of applying these task-similarity weights to the GP is shown in Figure 2.3b. Although, there is no additional information for g_1 directly, the information from the other functions is enough to improve the estimates of it.

Given a set of M functions (or tasks) $T = \{f_1, f_2, \ldots, f_M\}$, a matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^T$ of N D-dimensional training inputs with $\mathbf{x}_i \in R^{D\times 1}$ taken from functions $\mathbf{t} = (t_1, t_2, \ldots, t_N)^T$ with $t_i \in T$ and corresponding observed outputs $\mathbf{y} = (y_1, y_2, \ldots y_N)^T$. Given a matrix $\mathbf{X}_* = [\mathbf{x}_{*1}, \mathbf{x}_{*2}, \ldots, \mathbf{x}_{*L}]^T$ of L D-Dimensional test inputs. Let $k^f(f_p, f_q)$ be a function that returns the similarity of function f_p and f_q and thus $\mathbf{K}^f(\mathbf{t}_i, \mathbf{t}_j)$ be the task-similarity matrix, such that each element $\mathbf{K}^f_{pq}$ is calculated by $k^f(t_p, t_q)$, analogical to $\mathbf{K}$. How function k^f is defined will be discussed in Section 2.3.

Applying the task-similarity to the mean and the covariance predictions changes the equations for the single GP (2.4) and (2.5) as follows to make predictions for the function value of the l^{th} function for inputs $\mathbf{X}_*$ [7]:

$$\bar{\mathbf{f}}_{*l}(\mathbf{X}_*) = \left[\mathbf{K}^f(f_l, \mathbf{t})) \odot \mathbf{K}(\mathbf{X}_*, \mathbf{X})\right] \left[\mathbf{K}^f(\mathbf{t}, \mathbf{t})) \odot \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n \mathbf{I}\right]^{-1} \mathbf{y} \tag{2.6}$$

$$\text{cov}(\mathbf{f}_{*l}) = \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*) - \left[\mathbf{K}^f(f_l, \mathbf{t}) \odot \mathbf{K}(\mathbf{X}_*, \mathbf{X})\right] \left[\mathbf{K}^f(\mathbf{t}, \mathbf{t}) \odot \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n \mathbf{I}\right]^{-1} \left[\mathbf{K}^f(\mathbf{t}, f_l) \odot \mathbf{K}(\mathbf{X}, \mathbf{X}_*)\right] \tag{2.7}$$

where $\odot$ denotes the element-wise multiplication.

2.3 Hyper-parameter Optimization

Making good estimates using (multi-task) GPs depends on how good the hyper-parameters are adjusted. So they have to be optimized.

The values in the kernel function used by the standard GP are the hyper-parameters which need to be optimized. In the Squared Exponential Kernel (2.1) there are three hyper-parameters: l, σ_f and the noise σ_n.

These hyper-parameters are optimized by maximizing the marginal Log-Likelihood of the training data $\mathbf{X}$. The marginal Log-Likelihood is calculated by [13, p.19]:

$$\log p(\mathbf{y}|\mathbf{X}) = -\frac{1}{2}\mathbf{y}^T\mathbf{K}(\mathbf{X},\mathbf{X})^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K}(\mathbf{X},\mathbf{X})| - \frac{N}{2}\log(2\pi) \tag{2.8}$$

Where $\mathbf{X}$ is a matrix of N training data points, $\mathbf{y}$ is a vector of corresponding observed function values and $\mathbf{K}$ is the Gram matrix.

To calculate the Log-Likelihood the available data has first to be split up into a train and a test set, such that the parameters can be optimized on the train set and then be evaluated on the test set. If there is only little data available it is more data efficient to use a cross-validation approach for the optimization. This means that the data is first split up into J parts with each part containing the same amount of data points. These parts are called folds. In this case the objective for the optimization is the Log-Likelihood of one fold, given J-1 folds as training. In each optimization step each fold is once the test fold and the resulting Log-Likelihoods are summed. This value is then maximized.

The Log-Likelihood of observing the output $\mathbf{y}_*$ of the test data point $\mathbf{x}_*$ given the training data $(\mathbf{X},\mathbf{y})$ is computed by [13, p.116]:

$$\log p(y_*|\mathbf{x}_*,\mathbf{X},\mathbf{y}) = -\frac{1}{2}\log(\sigma^2) - \frac{(y_* - \bar{f}_*)^2}{2\sigma^2} - \log(2\pi) \tag{2.9}$$

Where $\bar{f}_*$ is the estimate calculated by (2.4) or (2.6) respectively and σ^2 is the variance calculated by (2.5) or (2.7) respectively.

The Log-Likelihood for the whole set of L test points $\mathbf{X}_*$ given the training data is calculated by:

$$LL(\mathbf{y}_*|\mathbf{X}_*,\mathbf{X},\mathbf{y}) = \sum_{i=1}^{L}\log p(y_i|\mathbf{x}_{*i},\mathbf{X}) \tag{2.10}$$

In multi-task GPs there are additional parameters. For the data kernel the parameters are the same as mentioned above. But for the task kernel function k^f (Section 2.2) additional parameters have to be learned. How these parameters look like depends on how function k^f is defined and therefore how the entries of the task-similarity matrix K^f are calculated. Two ways to define this function are discussed in the following sections.

2.3.1 Task-Similarity approach

The first approach to set the task-similarity matrix $\mathbf{K}^f$ is simply setting the values of the matrix directly. The additional parameters are in this case the entries of $\mathbf{K}^f$, i.e. the function k^f is defined such that $k^f(f_p,f_q) = \mathbf{K}_{pq}$. Since the matrix $\mathbf{K}^f$ is symmetric, the number of additional parameters is $(M(M-1))/2$, where M is the number of tasks. So the number of additional parameters is quadratic in the number of tasks.

This approach is used in [7], where the authors make a few assumptions that makes the calculation of this matrix easier. Like having the same inputs $\mathbf{X}$ for all functions and having noiseless observations (training inputs).

When optimizing the parameters, it has to be guaranteed that the resulting matrix $\mathbf{K}^f$ is positive definite (PD). This has to be ensured, since $\mathbf{K}^f$ is a covariance matrix, containing the covariances between the tasks. In this case every time the parameters are optimized, $\mathbf{K}^f$ has to be checked for positive-definiteness. If it is not PD it has to be manipulated, such that positive-definiteness is fulfilled. In [15] Higham et al. describe a method that produces the nearest Symmetric Positive Definite (nearest SPD) matrix to a non-PD square matrix. For the implementation a function is used provided by John D'Errico[1] which returns the mentioned nearest SPD matrix.

[1] nearestSPD - File Exchange - MATLAB Central: `http://www.mathworks.com/matlabcentral/fileexchange/42885-nearestspd` (August 2014)

2.3.2 Feature Based approach

Instead of directly optimizing the task similarities, also a kernel function for the tasks can be defined [9, 10]. This function calculates the similarity between two tasks. In this case the tasks (or functions) has to be represented by so called task-features. So the function k^f is defined as a kernel function (like the Squared Exponential Kernel (2.1)) calculating the covariance of two tasks, respectively their feature representations. Instead of calculating $k^f(f_p, f_q)$, the tasks are first projected into feature space by $\phi(f_i)$ and the covariance in this feature space is calculated. So the entries of $\mathbf{K}^f$ are calculated by $\mathbf{K}_{pq} = k^f(\phi(f_p), \phi(f_q))$.

The additional parameters in this case are the feature representations of the tasks, such that there are MD_ϕ additional parameters, where M is the number of tasks and D_ϕ the dimensionality of the feature space. So in this case the number of parameters is linear in the number of tasks.

The big advantage of using this approach compared with the task-similarity approach (Section 2.3.1) is that the resulting task-similarity matrix $\mathbf{K}^f$ is guaranteed to be a covariance matrix, i.e. it is positive definite, since its entries are calculated by a positive definite kernel function.

Preliminary experiments show that using two dimensional features is better than one dimension for the feature space. Even when the number of parameters doubles, the optimization gets less stuck in local minima.

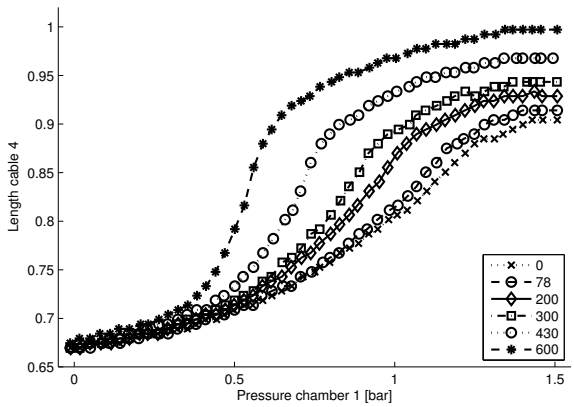
3 Experiments

To evaluate the described method a pneumatically driven, compliant robot is used. From this robot data is gained and then used for the computations. The results are compared with a standard GP method.

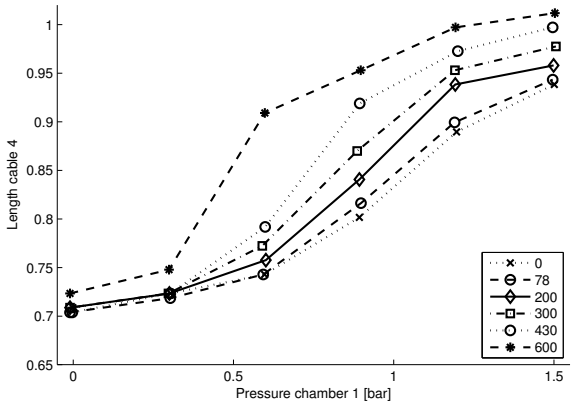
The robot that is used is the Robotino XT, as it is introduced in Chapter 1. One can send and receive data to and from the robot through the network. To interact with the robot an API and a ROS (Robot Operating System) interface is provided. A few functionalities had to be implemented manually for the ROS interface, like starting the compressor or closing the gripper. Through this ROS interface the pressures are sent to the robot and the state of the BHA is received.

Dataset

For training the GPs two data sets are gathered. For each of the data sets the robot has to lift different weights. In the following each weight to be lifted is called a task. The set of tasks for this experiment is $T = [0, 78, 200, 300, 430, 600]$, where each element of T specifies the weight of the load in gram. In the first data set the upper three chambers (chamber 4-6) are fixed to 0 bar pressure and the second and third chambers are fixed to 1.5 bar. To the first chamber pressures between 0 and 1.5 bar in steps of 0.03 bar are applied (see Figure 1.2 to see chamber numeration). So for each task there are 51 data points. In the following this is called the "small" data set (Figure 3.1a). The other data set is generated by changing the pressures in the lower three chambers (1-3), from 0 to 1.5 bar in steps of 0.3 bar. This results in a set of 216 data points per task. This is called the "big" data set. Figure 3.1b shows a plot of a slice of the data, where chamber 2 and 3 are fixed to 1.5 bar, as for the small data set. For the experiments only the cable length of cable 4 is considered, since preliminary tests showed that it gains the best results. Figure 3.1a shows that the data of this cable is nicely distributed for the different tasks. Cable 4 is the upper one which goes to the end effector (see Figure 1.2) so this is also the cable that is effected the most if the BHA bends to the ground, due to the weights it is lifting.



(a) Small Data Set



(b) Slice of Big Data Set, with pressure of chamber 2 and 3 fixed to 1.5 bar

Figure 3.1: Data. The numbers in the legend denote the weights of the objects that are lifted to get the data.

For the experiments it is assumed that there are several known or "previously seen" tasks and one new task. Previously seen means, that there is already a lot of data for this tasks. For the previously seen tasks all the data is used for training, since it should be simulated that there are already models for these tasks and for the new task just a few points are randomly picked. Whereas the number of data points for the new task is varied. The experiments should show which advantage the number of previously seen tasks and the amount of training data for the new task have. Also which benefit the multi-task GP approach has to single GPs.

The new task is the one with 430 grams in this case. This task is chosen, since it is no extreme case (empty or 600 grams) and it is most different to the other tasks, so it is the most interesting to observe. From the data of this task a

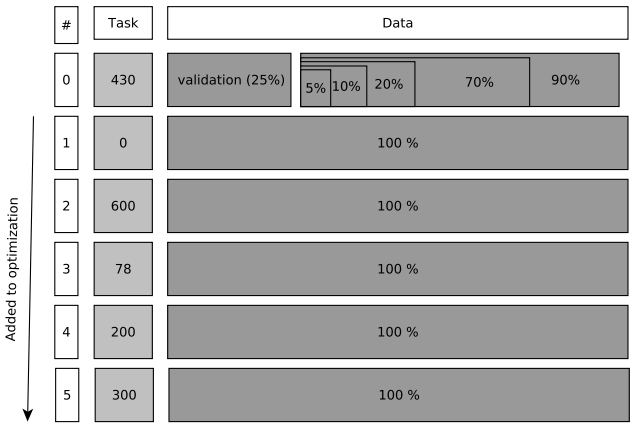


Figure 3.2: Illustration of data organization for the experiments. For training 5-90% of the data of task 430 and 100% of the other task is used

validation set with 25% of the data is extracted (Figure 3.3 (example for small data set)). This validation data is used to measure the quality of the learned models, by calculating the negative Log-Likelihood (nLL) and the Root Mean Squared Error (RMSE). From the 75% remaining data points of this task different amounts are used for training the GPs. For the experiments a set of A amounts is tested, with $A = \{5\%, 10\%, 20\%, 70\%, 90\%\}$. For the small data set this means a number of data points of 2 (5%), 4 (10%), 8 (20%), 27 (70%) and 34 (90%). As there are 51 data points available and 13 are taken for validation. Also different numbers of previously seen tasks are tested. The tasks are added in the order $0, 600, 78, 200, 300$. So first the extreme cases and then ascending weights are added. Figure 3.2 shows how the data is organized.

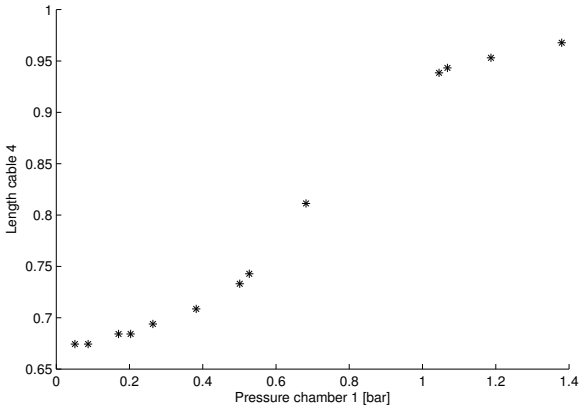
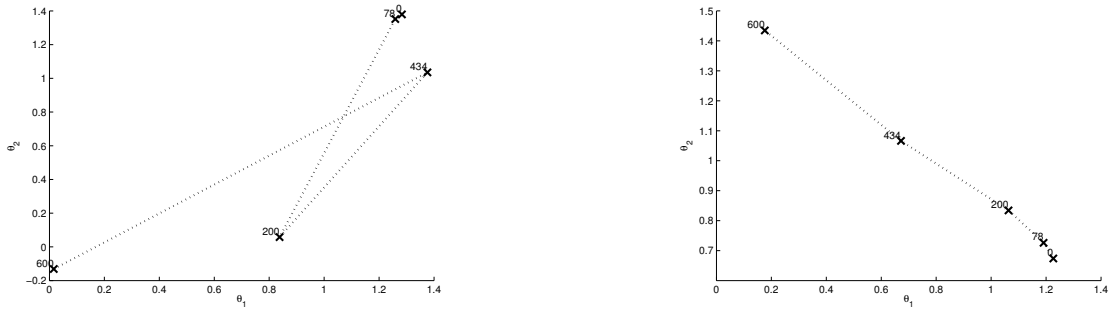


Figure 3.3: Extracted validation data of small data set

3.1 Forward Model

To test the multi-task GP approach for learning the forward dynamics several settings are tested. As mentioned above different amounts of training points for the new task and different number of previously seen tasks are evaluated. Also two objectives for the hyper-parameter optimization are investigated (Section 2.3). One is maximizing the marginal Log-Likelihood, where first the data of every task is split up into 10% test data and 90% training data. Then the log-likelihood of the training data is maximized and the model is evaluated on the test data. After 10 repetitions, the model with the highest log-likelihood on test data is chosen. The other is the cross-validation ML approach. The available data is split up into J folds (in this case J = 4). On these J folds, J times the log-likelihood on one set as test set given J-1 sets as train set is maximized. There are 10 iterations, after which the model with the highest cumulative log-likelihood is chosen.

The optimization has to be repeated 10 times, since the result highly depends on the starting values of the similarities respectively the features. At some starting points the optimization gets stuck in local minima. An example of two learned models for the feature-based approach is shown in Figure 3.4. It shows two resulting feature sets of the optimization, given 4 previously seen tasks and 20% training data of the new task. The best result is a monotone function of the features since task 0 should be more similar to task 78 then to task 200 and so on. Figure 3.4a shows a bad example for features. The learned task-features are spread over the space and according to this task 78 is more similar to 430 than to 200, as Figure 3.4a shows a local minimum found by the optimization. Compared to Figure 3.4b where all the features are ordered according to their weights as desired.



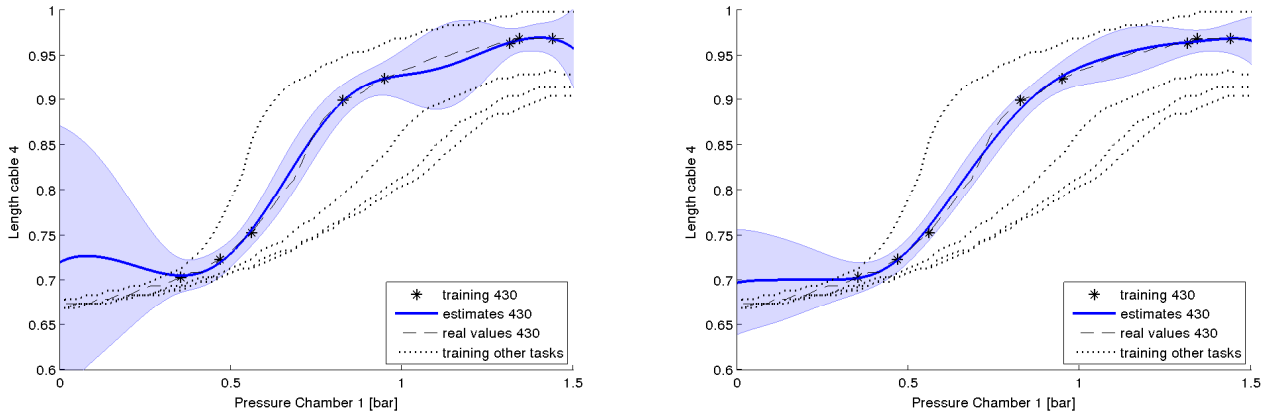
(a) Bad example **(b)** Good example

Figure 3.4: Example for learned 2-dimensional task-features of two different optimization trials. The trials are done with different starting values for the features

Compare nearest SPD and feature-based approach

First different approaches to model the task-similarities are compared. As mentioned in Section 2.3 there are different ways to look at the task-similarities. Two approaches are tested, the "nearest SPD" (Symmetric Positive Definite) approach (Section 2.3.1 (Task-similarity approach)) and the "feature-based" approach (Section 2.3.2). During the experiments it shows that nearest SPD is much more complex in computation, since it needs more time for the optimization. This is caused by more parameters that has to be optimized and also by more calculation steps in the optimization. In each step the learned parameters of the task-similarity matrix do not form a positive definite matrix, the matrix has to be updated by using the Cholesky decomposition and computing its eigenvalues several times.

Figure 3.5 shows the results for 2 previously seen tasks and 20% of training data for the new task. The dashed line are the real values of the cable lengths for the new task, the dotted lines are the other known tasks, the crosses are the training values for the new task and the blue area is two times the standard deviation of the estimates. Both parameter sets are learned with the cross-validation maximum likelihood method.



(a) nearest SPD - nLL on validation set: -36.5087

(b) Feature-based - nLL on validation set: -42.4129

Figure 3.5: Estimates of feature-based and nearest SPD approach on small data set learned with cross-validation objective. The blue area denotes 2 times the standard deviation.

The results in Figure 3.5 show that both approaches are producing similar results, whereas the ones of the feature-based approach are better. The feature-based approach needs less parameters and its learned task-similarity matrix is guaranteed to be is positive definite. The task-similarity matrix is positive definite since its entries are calculated by a positive definite kernel function. Figure 3.6 shows the nLL on the validation set with the learned models of the two approaches given 20% of training points for the new task. It shows that the feature-based approach outperforms the nearest SPD approach for all numbers of previously seen tasks. Further the feature-based approach has a smaller parameter space so its results are more robust to different starting values. Since the feature-based approach has better performance and gains better results, only this approach is used in the following experiments.

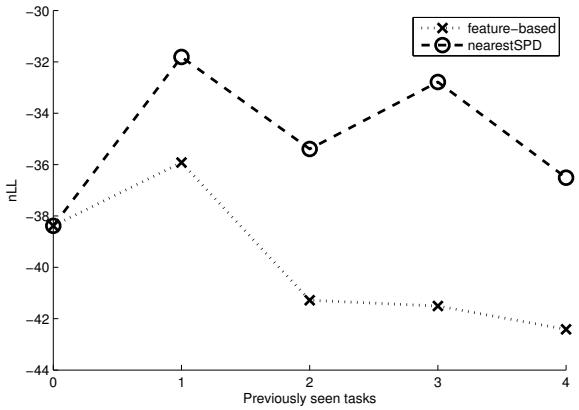
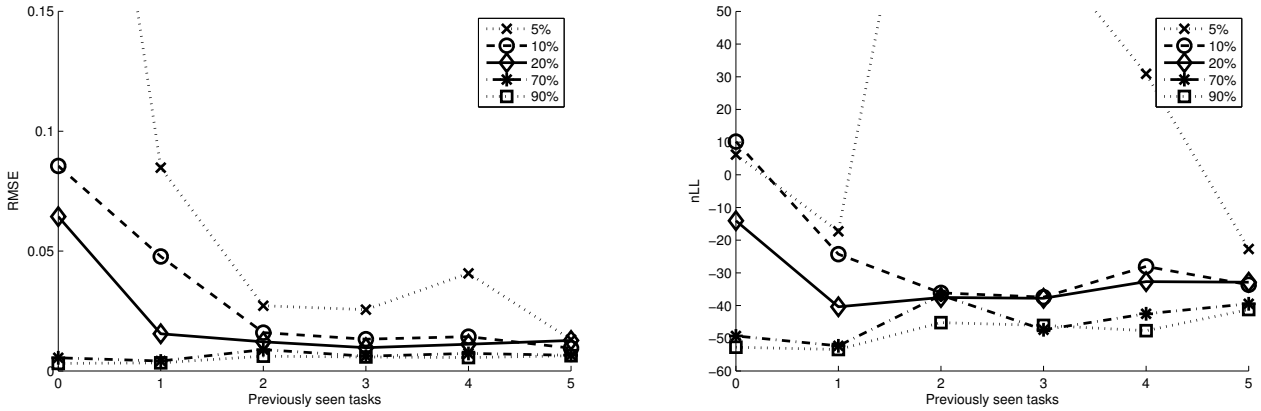


Figure 3.6: Comparison of nLL of feature-based and nearest SPD approach on small data set using the cross-validation objective for optimization and 20% of data of the new task as training

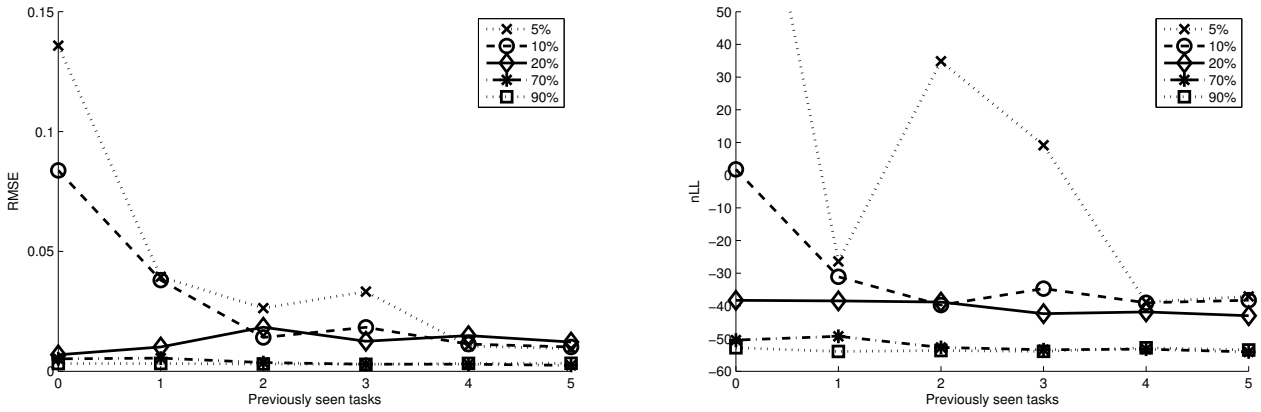
Compare maximizing marginal log-likelihood and cross-validation objective

As mentioned above there are different ways that can be used to optimize the parameters. In this experiment the two objectives are tested and compared. The marginal log-likelihood and the cross-validation objective. Both objectives are used to optimize the parameters and calculate the errors on the evaluation set. Figure 3.7 shows the results for the marginal log-likelihood objective and 3.8 for the cross-validation. On the x-axis is the number the previously seen tasks. The percentages denote the amount of training points for the new task. In Figure 3.7 the ML objective is used for optimization on a generated test set. Figure 3.8 shows the results for the cross-validation objective.



(a) RMSE **(b)** nLL

Figure 3.7: Results obtained by maximizing the marginal likelihood on small data set. The percentages denote the amount of data of task 430 that is used for training



(a) RMSE **(b)** nLL

Figure 3.8: Results obtained by cross validating the small data set. The percentages denote the amount of data of task 430 that is used for training

Comparing Figure 3.7 and Figure 3.8 shows that the cross-validation gains better results, as the errors in Figure 3.8 are for all cases lower than in Figure 3.7. Especially for the cases with 5-20% of training data for the new task. Using the cross-validation approach has the advantage that the result is less depending on the test and train set that are chosen. When optimizing marginal log-likelihood without cross-validation one has to split up the data set into train and test sets. Whereas the log-likelihood is only optimized on this chosen train set, the test set is then used to decide which model to use, since the optimization is repeated 10 times. The resulting model highly depends on the data points that are in the train set. For the 5% case there are only two data points for the new task, that means that there is one point in the test set and one in the training. The consequence of this can be seen in Figure 3.7b for 5%, where the error for 2, 3 and 4 previously seen tasks is much higher than using cross-validation (Figure 3.8b). Cross-validation avoids the dependency on the choice of the train and test set by using different subsets for train and test, as each subset is used once as test set. This is more data efficient then just using one particular set for test. Since the small data set is used where only 51 data points per task are available, the cross-validation objective has to be used for optimization.

Compare different number of tasks and different amount of training points of new task

To evaluate the advantage of multi-task GPs against normal GPs, following settings are analyzed. Different amount of training data for the new task and different numbers of previously seen tasks are compared. The standard GP only on the data of the new task means in this setting that there are zero previously seen tasks. As suggested in [8] also the case

is evaluated when all data of all tasks is put into one single GP.

Figure 3.9 shows the influence of the number of previously seen tasks on the estimation, given 5% of the data of the new task for training. So there are only 2 data points as training for the new task, as the only information for this. It shows that the estimates are getting better the more tasks are added. With 2 and 3 previously seen tasks the estimates seem to adapt too much to the upper (the 600) task, but according to the position of the two data points this is reasonable. Figure 3.9 also shows the reason for the high values in Figure 3.8 for 5% and 2 and 3 previously seen tasks. The estimates for the new task are adapting too much to the 600 task so that the real values are out of the range of two times the standard deviation, which increases the negative Log-Likelihood. With more than 3 tasks it gets better, since the optimization adapts the parameters such that the new task is more similar to the lower task. The optimization estimates the tasks to be too similar, adding more data points to the new tasks tackles this problem. As Figure 3.8 shows, with only two more data points (10% of training data for the new task) the negative Log-Likelihood gets more stable, i.e. the task-similarities are estimated more accurate.

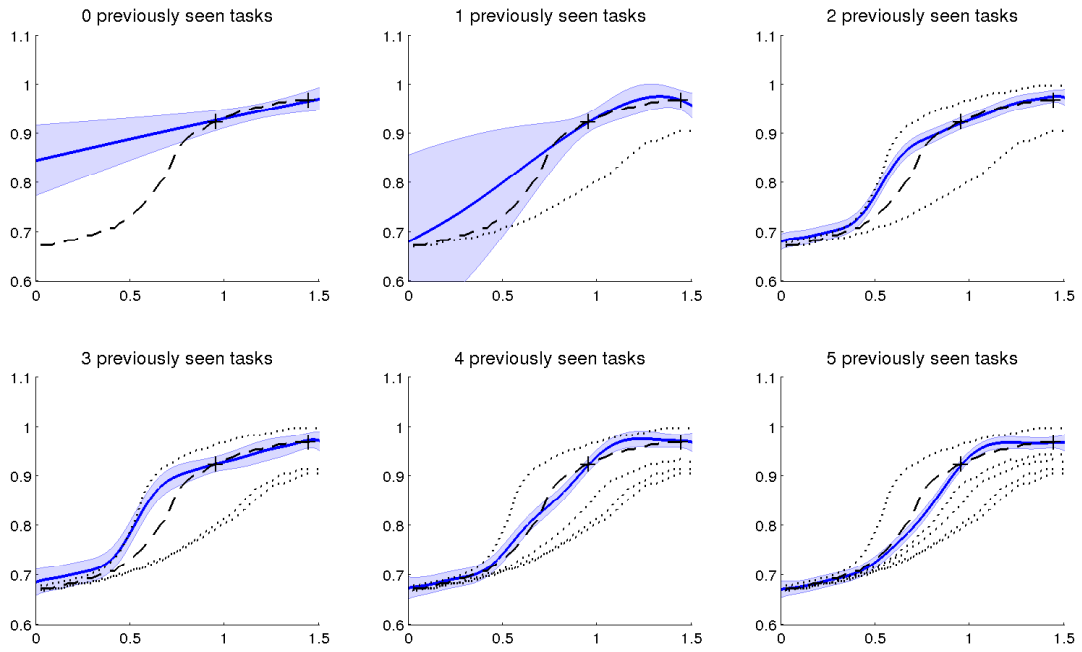


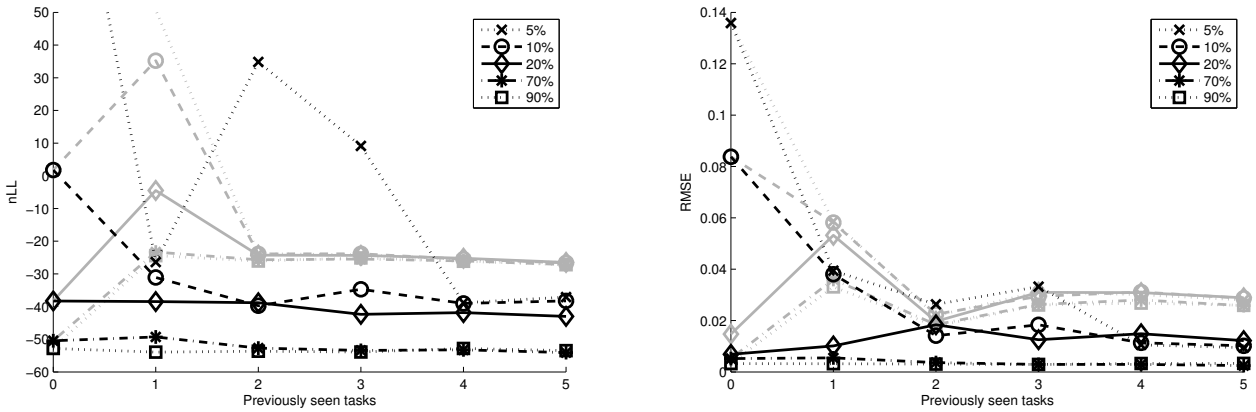
Figure 3.9: Predictions for the new task using 5% of the new tasks data. In each subplot a number of previously seen tasks is displayed. Beginning with zero previously seen tasks on the upper left and ending with five on the lower left. The x-axes denote the pressure that is applied to chamber 1 and the y-axes the length of cable 4. The dashed lines is the real function of the new task, the dotted lines are the real functions of the added tasks (which are used for training), the markers are the training points of the new task.

Figure 3.8 and Figure 3.9 show the advantage of multi-task GPs to standard GPs (the case of 0 previously seen tasks). If there is only little data, as in this case only two data points, the multi-task GP outperforms the single GP as soon as one other task is available. They also show that having many data points for the new task, the multi-task GP does not gain any improvement in estimations, it sometimes worsen the results, since the estimates are also adapted to the other tasks. In this setting there is a improvement using multi-task GPs when using 10% or less data of the new task for training. With 20% or more the multi-task GPs do not improve the results remarkably.

Compare multi-task GP to one single GP on all data

To show that there is an advantage in using multi-task GPs instead of just using one single GP for all the data, this case is also evaluated. Figure 3.10 shows the results with underlying results for the case that all data points are in one single GP. From the point where 3 tasks are added the amount of training for the new task does not matter, since there are enough other data points, so these do not influence the result too much. It also shows that the multi-task GP always outperforms the one GP. Only in two cases the one GP is better than the multi-task GP, which are the mentioned cases

for 5% training data of the new task and 2 and 3 previously seen tasks, where the task features are optimized such that the similarity to task 600 is very high so it adapts too much to this task. The single GP performs in these cases better, since the two previously seen tasks are 0 and 600, so the 430 task should be somewhere in the middle of these two tasks, which improves the estimates of the single GP.

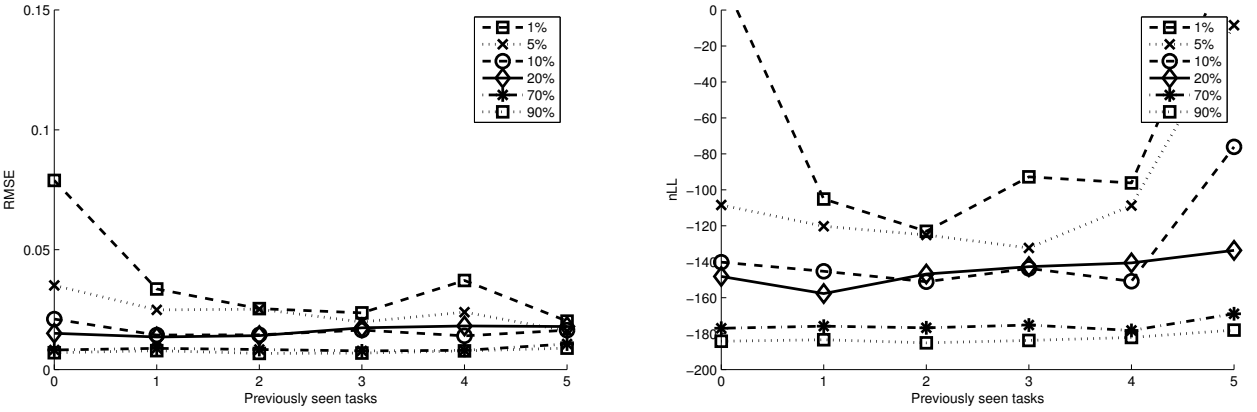


(a) nLL **(b)** RMSE

Figure 3.10: Results of multi-task GP compared to one GP on all data. The underlying grayed data is for the one GP

Big data set

To evaluate the method further it is applied to the big data set containing the variation of three of the chambers, so that the data inputs are three dimensional. The output is still one dimensional (only cable 4 is investigated). Figure 3.11 shows the results for the big data set using cross-validation and the feature-based task-similarity approach.



(a) RMSE **(b)** nLL

Figure 3.11: Results on big data set. The percentage denotes the amount of training data of the 430 task. The hyper-parameters are optimized using cross-validation and the feature-based approach for the task-similarities

It shows that also on the big data set the results are good. As on the small data set there is only an improvement using the multi-task GPs if there is a small amount of training points for the new task. In this setting 1% of training data means 2 data points and 5% 8 data points.

Adding the fifth task worsens the negative Log-Likelihood, for 1%, 5% and 10%, in these cases the negative Log-Likelihood of the evaluation data is even worse than with a single GP (0 previously seen tasks). The learned parameter show, that with the last task added, the optimization learns high similarities (almost 1) for the tasks, this causes the variance to get low and this is why the nLL goes up but the RMSE is stable or gets better. So the optimization gets stuck in a local minimum. This illustrates again the high dependence on the data.

3.2 Inverse Model

Also the inverse model is learned using multi-task GPs. To invert the data, the input and the output of the data is switched. So given a certain cable length a pressure that should be applied is estimated. Respectively given a state, an action should be estimated. Figure 3.12 shows the inverted data.

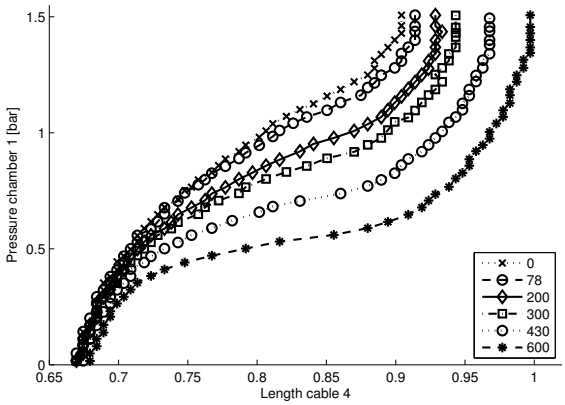


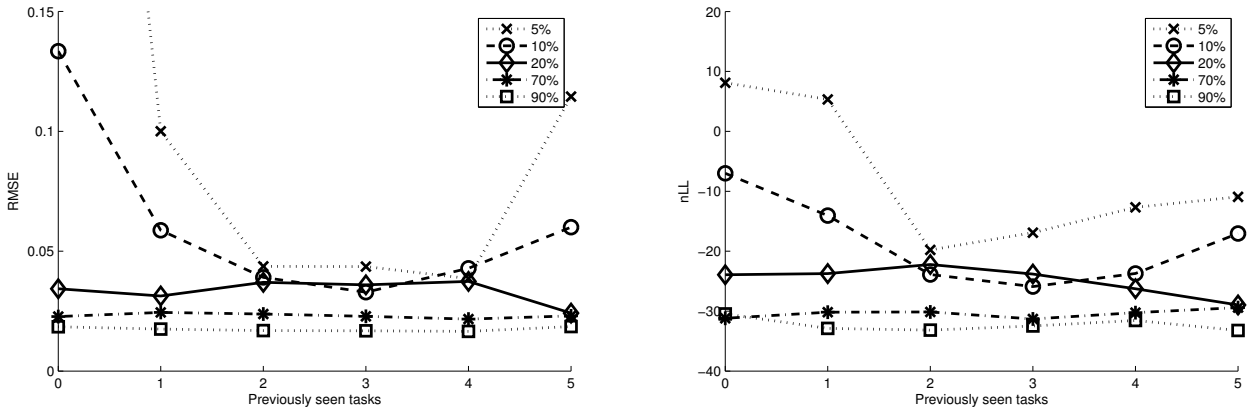
Figure 3.12: Inverted data. The numbers in the legend denote the weights of the objects that are lifted to get the data.

The evaluation of the inverse model is analogical to the forward model. All the experiments are done using the cross-validation objective (Section 2.3) for optimization and the feature-based approach (Section 2.3.2) for the task-similarities. The inverted data of the small data set is used as input and just the length of cable 4 is investigated. The experiments are done using the same set of amounts for the new (430 grams) task and the "previously seen tasks" are added in the same order as in the forward model case (see Figure 3.2).

Two different approaches to learn the inverse model are evaluated. The first approach is to optimize the hyper-parameters analogically to the forward case, but on the inverted data. In this case that means to optimize the hyper-parameters the probability of applying a control input to the robot given a state that should be reached is maximized ($p(a|s)$). However, maximizing a probability of applying an control input is not very intuitive, since there is not really a distribution of applying an action. A more common way is to maximize the probability of reaching a state given an applied control input ($p(s|a)$), which is what is optimized in the forward case. So the second approach that is evaluated is to first learn the task-features on the forward data and subsequently optimize the hyper-parameters for the data-kernel on the inverted data. This is also inspired by the MOSAIC framework [16], which first evaluates the best forward model description of the data and then the corresponding inverse model is used for control. In the following the first approach is called the inverse-inverse case (inv-inv) and the second the forward-inverse case (fwd-inv).

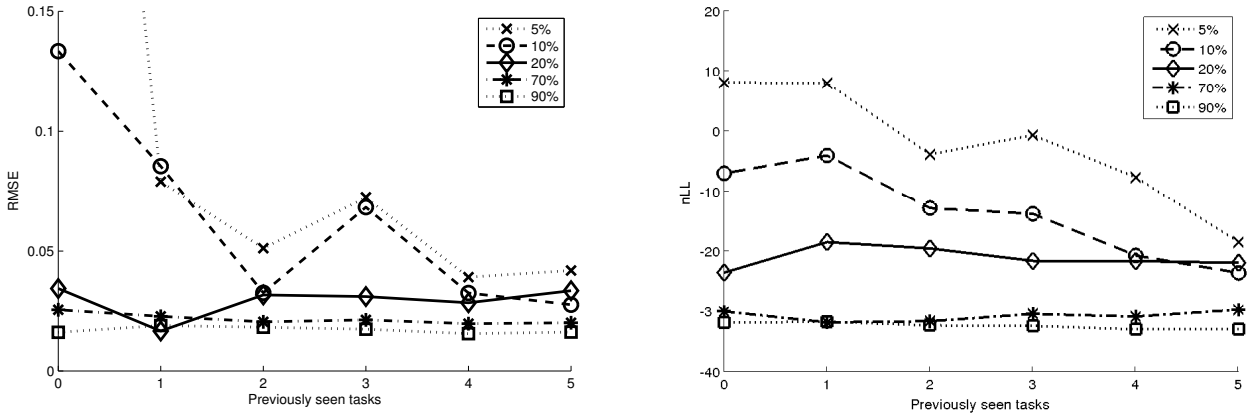
Figure 3.14 displays the errors on the validation data for the fwd-inv case and Figure 3.13 for the inv-inv case.

It shows that the fwd-inv case is not improving the results. Only in the cases with 5% and 10% of training data for the new task and 5 tasks added. For these cases the inv-inv approach gets stuck into local minima and the error increases. In general the inverse model learning has higher errors than the forward case. This is caused by the data. The data on the right (in the interval [0.85, 1]) has a very high gradient for all tasks. Figure 3.15 shows the inverted data, but for the 430 task only the training data for the 5% case, i.e. the two data points that are used for training. It shows that for the 430 task and the tasks 0 and 78 no training points are on the same x-value. So calculating their similarity results in very low values. Adding the 200 and the 300 task (which are the forth and fifth tasks added, as to see in Figure 3.2) changes this and so the optimization can get stuck in local minima.



(a) RMSE **(b)** nLL

Figure 3.13: Errors on validation data for inv-inv case. All the hyper-parameters are optimized on the inverted data. The percentages denote the amount of training data from task 430 (cf. Figure 3.2)



(a) RMSE **(b)** nLL

Figure 3.14: Errors on validation data for fwd-inv case. The task-features are first optimized using the forward data and then applied to the inverted data. The percentages denote the amount of training data from task 430 (cf. Figure 3.2)

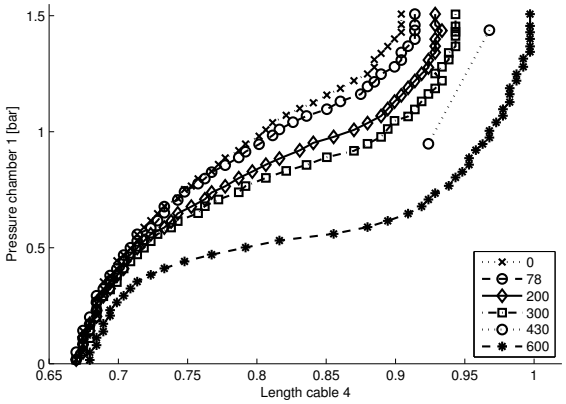


Figure 3.15: Inverted data with 5% of 430 tasks data

4 Conclusion

In this thesis multi-task Gaussian Processes (multi-task GPs) are used to tackle the problem of learning generalizable models for compliant robots performing different tasks. The concept of task-similarities and multi-task GPs is presented. A multi-task GP framework is implemented and applied to data gathered from a real compliant robot. The results are evaluated using different amounts of training points for a new weight to lift (or new task) and different number of tasks with a high amount of data. The resulting estimates and their errors are compared to single GP approaches.

Two objectives for the optimization are compared, the marginal log-likelihood and the cross-validation Log-Likelihood objectives, whereas the cross-validation objective gains better results. Different approaches to model the task-similarities are investigated, one estimates the similarities directly from the data and the other defines task-features and calculates the similarities of the tasks in feature-space. In the first approach it has to be checked whether the resulting task-similarity matrix is positive definite, which causes additional computation time. This can be avoided using the second approach, where the task-similarity matrix is computed by a kernel function which produces a positive definite matrix. Further, the second approach has fewer parameters to optimize and gains better results, so this approach is concluded to be better. The effect of different amounts of training points for a new task and different number of already known tasks are analyzed by learning the forward dynamics model for one and for three actuators.

The results show that multi-task GPs are helpful if there is much data for a set of tasks and estimates for a new task have to be made, whereas for this new task only a very small amount of data is available. If the amount of data for the new task is already high, single GPs perform equally good or even better than multi-task GPs. In this case due to computation time single GPs should be preferred, since the covariance matrix that has to be inverted is $MN \times MN$ where M denotes the number of tasks and N the number of data points per task. In single GPs just a $N \times N$ matrix has to be inverted.

For the inverse dynamics problem two approaches are investigated, one works analogical to the forward dynamics model learning approach, just on inverted data, where the x and y values are swapped. The other approach first computes the task-features on the forward data and uses them for learning the inverse model. It shows that for the data that is used in this thesis, the second approach does not improve the results. But, as for the forward case, the multi-task GPs are improving the results in comparison to single GPs, when there is only a low amount of data for a new task.

Future Work

Evaluating the model on the real robot is the next milestone, since this was not possible due to time constraints. Therefore, the method has to be scaled up to work for all the actuators and all the cables of the robot. Especially applying the inverse dynamics model for control to the robot is interesting to focus on. For this the inverse dynamics learning has to be further investigated. Also applying the method to data from other robots to show how it generalizes to different applications, would be interesting.

A case that is not analyzed in this thesis, is to just learn a model for all the data of the previously seen tasks and only optimize the task-features of the new task, leaving the rest as it is. This should improve computation time, since the number of newly learned parameters reduces to two.

Bibliography

[1] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. John Wiley & Sons New York, 2006.

[2] J. Kober and J. Peters, "Reinforcement learning in robotics: A survey," in *Reinforcement Learning* (M. Wiering and M. van Otterlo, eds.), vol. 12 of *Adaptation, Learning, and Optimization*, pp. 579–610, Springer Berlin Heidelberg, 2012.

[3] C. Escande, P. M. Pathak, R. Merzouki, and V. Coelen, "Modelling of multisection bionic manipulator: Application to RobotinoXT," in *Robotics and Biomimetics (ROBIO)*, pp. 92–97, IEEE, 2011.

[4] T. Mahl, A. Hildebrandt, and O. Sawodny, "Forward kinematics of a compliant pneumatically actuated redundant manipulator," in *Industrial Electronics and Applications (ICIEA), 2012 7th IEEE Conference on*, pp. 1267–1273, July 2012.

[5] B. Bischoff, D. Nguyen-Tuong, A. van Hoof, H. McHutchon, C. Rasmussen, A. Knoll, J. Peters, and M. Deisenroth, "Policy search for learning robot control using sparse data," in *Proceedings of 2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[6] M. Rolf and J. Steil, "Efficient exploratory learning of inverse kinematics on a bionic elephant trunk," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 25, pp. 1147–1160, June 2014.

[7] E. V. Bonilla, K. M. A. Chai, and C. K. Williams, "Multi-task gaussian process prediction.," in *Neural Information Processing Systems (NIPS)*, vol. 20, pp. 153–160, 2007.

[8] K. M. A. Chai, C. K. Williams, S. Klanke, and S. Vijayakumar, "Multi-task gaussian process learning of robot inverse dynamics.," in *Neural Information Processing Systems (NIPS)*, pp. 265–272, 2008.

[9] E. V. Bonilla, F. V. Agakov, and C. Williams, "Kernel multi-task learning using task-specific features," in *International Conference on Artificial Intelligence and Statistics*, pp. 43–50, 2007.

[10] K. Yu, V. Tresp, and A. Schwaighofer, "Learning gaussian processes from multiple tasks," in *Proceedings of the 22nd international conference on Machine learning*, pp. 1012–1019, ACM, 2005.

[11] J. R. Flanagan, S. King, D. M. Wolpert, and R. S. Johansson, "Sensorimotor prediction and memory in object manipulation.," *Canadian Journal of Experimental Psychology/Revue canadienne de psychologie expérimentale*, vol. 55, no. 2, p. 87, 2001.

[12] C. E. Rasmussen, *Evaluation of Gaussian processes and other methods for non-linear regression*. PhD thesis, University of Toronto, 1996.

[13] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[14] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer Science + Buisness Media, 2006.

[15] N. J. Higham, "Computing a nearest symmetric positive semidefinite matrix," *Linear Algebra and its Applications*, vol. 103, pp. 103 – 118, 1988.

[16] D. M. Wolpert and M. Kawato, "Multiple paired forward and inverse models for motor control," *Neural Networks*, vol. 11, no. 7, pp. 1317–1329, 1998.