Model Learning for Probabilistic Movement Primitives

Modell Lernen für Probabilistische Movement Primitives
Bachelor-Thesis von Pascal Klink aus Weinheim
Tag der Einreichung:

1. Gutachten: Prof. Dr. techn. Gerhard Neumann
2. Gutachten: Dipl.-Ing. Alexandros Paraschos

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Model Learning for Probabilistic Movement Primitives
Modell Lernen für Probabilistische Movement Primitives

Vorgelegte Bachelor-Thesis von Pascal Klink aus Weinheim

1. Gutachten: Prof. Dr. techn. Gerhard Neumann
2. Gutachten: Dipl.-Ing. Alexandros Paraschos

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter 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.

Darmstadt, den 19. Oktober 2016

_____

(Pascal Klink)

Abstract

Probabilistic Movement Primitives (ProMPs) provide a framework for representing distributions over robot movement trajectories, combining or altering those distributions and deriving motor signals which lead to a reproduction of the encoded movements. In order to derive those signals for a given robot, a linear model of the robot dynamics is required. If the robot dynamics are known, such a linear model can be obtained by linearizing the non-linear dynamics for the current joint configuration of the robot. If the robot dynamics are unknown, one can try to derive them from the movement data with which the ProMP was created. This thesis presents and evaluates different Machine Learning methods which can be used to learn linear models for calculating motor signals of ProMPs as well as approaches to facilitate the reproduction of trajectory distributions despite the presence of errors in the learned models.

Zusammenfassung

Probabilistic Movement Primitives (ProMPs) bieten die Möglichkeit Wahrscheinlichkeitsverteilungen über Bewegungstrajektorien von Roboterbewegungen darzustellen, diese zu manipulieren oder zu kombinieren und Motorsignale herzuleiten, die diese Wahrscheinlichkeitsverteilungen reproduzieren. Um die erwähnten Signale für einen Roboter herzuleiten, wird ein lineares Model von dessen Dynamik benötigt. Wenn die entsprechenden, zumeist nichtlinearen, Dynamikgleichungen bekannt sind, kann das lineare Model durch Linearisierung der Dynamikgleichungen für den aktuellen Roboterzustand erhalten werden. Wenn die Dynamikgleichungen jedoch unbekannt sind, kann man versuchen, diese aus den Bewegungsdaten herzuleiten, mit denen die ProMP erstellt wurde. In dieser Arbeit werden verschiedene Machine Learning Methoden zum Lernen von linearen Modellen für die Berechnung von Motorsignalen für ProMPs sowie Ansätze, um die Reproduktion von Wahrscheinlichkeitsverteilungen über Trajektorien trotz Fehlern in den gelernten Modellen zu ermöglichen, vorgestellt und evaluiert.

Acknowledgments

I would like to thank my supervisor Alexandros Paraschos for his advice, support and patience throughout the last semester. Furthermore, I want to give thanks to Philipp Becker for the amusing and prosperous discussions about my thesis topic from which I always drew new motivation for my work on it. Finally, I thank my family for always supporting me during my studies in the last years.

Contents

Figures and Tables

List of Figures

Abbreviations, Symbols and Operators

List of Abbreviations

Notation	Description
DMP	Dynamic Movement Primitive
MGD	Multivariate Gaussian Distribution
PDE	Probability Density Estimation
ProMP	Probabilistic Movement Primitive

1 Introduction

According to the Robotics Institute of America, a "robot is a reprogrammable multifunctional manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks." Robotics is the engineering field which is concerned with designing robots for mentioned tasks. As already pointed out in the definition, designing a robot often includes the explicit programming of the robot motions necessary to solve a task. In recent history, a new field referred to as Robot Learning emerged with the goal of enabling robots to autonomously learn a task instead of being programmed for it by combining the fields of Robotics and Machine Learning.

"Machine Learning can be broadly defined as computational methods using experience to improve performance or to make accurate predictions." [1] In [2], the author describes the very similar if not identical field of Pattern Recognition as a science that "[...] is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying [...] data into different categories." Both definitions capture the idea to use experience, which comes in the form of data of some kind, to be able optimize the performance on a certain task.

The field of Machine Learning comprises a vast number of methods to achieve this optimization. In recent history, different methods could be successfully applied to problems of various fields. One example is the application of so called deep neural networks in combination with Monte Carlo Tree Search to the board Game Go. [3] The resulting setup was the first ever to repeatedly beat professional Go players.

In the field of Robot Learning, Machine Learning methods are for example employed in so called Reinforcement Learning, where robots find optimal ways for solving a task by trial-and-error. [4] Especially relevant for this thesis is a branch of Robot Learning called Model Learning, where, among others, models for dynamics and kinematics are learned from movement data of robots. [5] Typically, a combination of methods from different branches lead to successful examples of Robot Learning, such as in [6], where a combination of Model- and Reinforcement Learning was used to create a helicopter controller that could accomplish an aerobatic maneuver called inverted flight. To achieve this result, model learning was used to find a forward dynamics model allowing the creation of a simulator for the helicopter. In this simulator, it was possible to learn the desired control law by Reinforcement Learning without the risk of destroying the helicopter due to a suboptimal control law.

Besides the Reinforcement Learning algorithm used in the above example, there exist newer approaches, such as the one presented in [7], which do not learn a control law but use parameterized representations of robot movements. The parameters which define the movement trajectory are then optimized to find the behavior that is optimal for the current task. Typically, the initial movement representation is bootstrapped using demonstration data. Such demonstration data can, e.g. be generated by teleoperation or kinesthetic teaching of the robot.

Multiple frameworks for creating parameterized movement representations exist, such as Dynamic Movement Primitives (DMPs) [8] or ProMPs [9]. For this thesis, especially ProMPs are of interest, where in contrast to DMPs, distributions over movements instead of only one movement trajectory can be expressed.

In order to create motor signals which will lead to the execution of the represented movement, one can for example use linear control laws to follow the target movement encoded by the movement primitives. However, if such linear controllers should be precise, they often become very stiff and with that dangerous for nearby people. This often prevents robots to be used for assisting humans in a task. Probabilistic Movement Primitives allow for a deduction of the necessary motor signals when a linear model of the robot dynamics is known. Such motor signals based on the knowledge of the dynamics model circumvent

the problem of the robot becoming stiff and dangerous. However, dynamic models are hard to retrieve for complex robots and hence it is desirable to learn those models from demonstration data instead of specifying it by hand.
Being able to successfully use learned dynamic models for generating motor signals of Probabilistic Movement Primitives would allow for a completely data-driven way of executing a movement representation while not limiting the usage of robots due to a control law which leads to a stiff and dangerous robot. Since also Reinforcement Learning methods employ such movement representations, positive results of this thesis would also allow for a broader application of movements learned by Reinforcement Learning.
This thesis will first introduce fundamental concepts in Machine Learning. Following this, a detailed introduction to the employed learning methods for Model Learning and Probabilistic Movement Primitives will be given. Based on this, the setup which was used to evaluate the learning methods will be described. Finally, the results will be reported and discussed.

2 Background

For the purpose of this thesis, Machine Learning will be interpreted as the problem of learning a mapping $f : I \times \Theta \mapsto O$ from inputs $x \in I$ to outputs $y \in O$ by finding appropriate parameters $\theta \in \Theta$. To do that, a set $\mathscr{D}$ containing sample data and some function $J(\mathscr{D}, \theta)$ that assesses the performance of f with the current parameters θ on the data set $\mathscr{D}$ is used. Commonly, the parameters are then retrieved by computing either $\underset{\theta}{\text{argmax}}\, J(\mathscr{D}, \theta)$ or $\underset{\theta}{\text{argmin}}\, J(\mathscr{D}, \theta)$. J is referred to as the reward or cost function, depending on whether maximizing or minimizing J leads to the desired parameters for f.
Depending on the structure of the sample data $\mathscr{D}$, the described learning problems can be divided into multiple categories. In this thesis, especially methods for solving so called Supervised Learning problems will be of interest. For Supervised Learning problems, the set of sample data contains inputs and observed outputs, i.e. $\mathscr{D} \subseteq I \times O$. An example for such a Supervised Learning problem would be the task of finding a relationship between the day of year and the power consumption of a city. One could model this task as a Supervised Learning problem by defining the input space to be the three hundred sixty-five days of a year, i.e. $I = \{1, \ldots, 365\}$, and the output space to be the real numbers which represent the consumed kWh of the city at a certain day, i.e. $O = \mathbb{R}$.

2.1 The Gaussian Distribution

Many of the Machine Learning methods which will be presented use the idea of maximizing the probability of observing the sample data $\mathscr{D}$ in order to find parameters $\boldsymbol{\theta}$. One probability distribution which is commonly employed in this optimization procedure is the so called Multivariate Gaussian Distribution (MGD)

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right).$$

A MGD is a probability distribution over d-dimensional real vectors $\mathbf{x} \in \mathbb{R}^d$ which is characterized by its mean $\boldsymbol{\mu} \in \mathbb{R}^d$ and covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$. The MGD is unimodal with mode $\boldsymbol{\mu}$, descriptively meaning that vectors which are similar to the mean of the distribution have a higher probability density than vectors which are very different from the mean.

Leaving mathematical sophistication aside, one could say that the covariance matrix $\boldsymbol{\Sigma}$ specifies how fast the probability density assigned to a vector decreases for increasing distance to the mean. The entries on the diagonal of $\boldsymbol{\Sigma}$ are referred to as the variances of the single dimensions σ_i^2 with $i = 1, \ldots, d$ and denote the pace of described decrease individually for each dimension. The entries on the off diagonal are the so called covariances and describe how a change in one dimension of the vectors is correlated with a change in another dimension. Looking at the formula of the MGD, the product $(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})$ should only be zero for $\mathbf{x} = \boldsymbol{\mu}$ (i.e. $\mathbf{x} - \boldsymbol{\mu} = \mathbf{0}$) and greater than zero for all other values of $\mathbf{x}$. Because of that, the covariance matrix needs to be positive definite and since the covariance between two dimensions a and b is denoted at indexes $\boldsymbol{\Sigma}_{a,b}$ and $\boldsymbol{\Sigma}_{b,a}$, it also needs to be symmetric. Figure 2.1 shows two two-dimensional MGDs for different covariance matrices.

2.1.1 Marginalization of Gaussian Distributions

In the following sections, an operation called marginalization will often occur. Given two probability distributions $p(\mathbf{x}|\mathbf{y})$ and $p(\mathbf{y})$, the integral $\int p(\mathbf{x}|\mathbf{y})p(\mathbf{y})\,\mathrm{d}\mathbf{y}$ is called the marginal. The name arises since

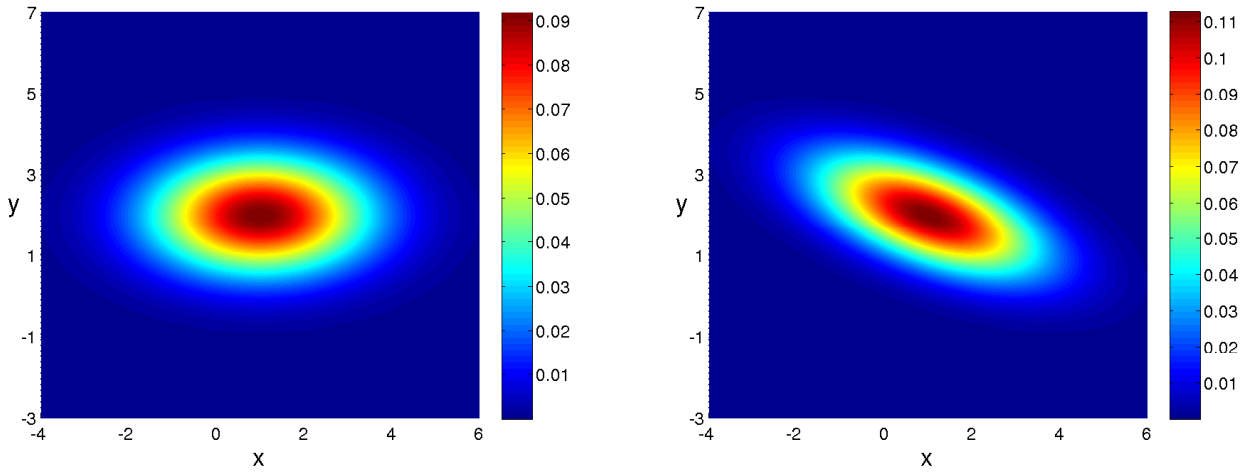

Figure 2.1.: The two plots show two two-dimensional MGDs both with mean $\boldsymbol{\mu} = [1\ 2]^T$. For the MGD in the left plot, the variances were set to $\sigma_1^2 = 3$ and $\sigma_2^2 = 1$ while leaving the covariance between the two dimensions at 0. One can easily see that this leads to a slower decrease of the probability density for increasing distance from $\boldsymbol{\mu}$ in the x- than in the y-dimension. In the right plot a covariance of -1 between the dimensions was added. Speaking descriptively, this rotates the distribution as the negative covariance encodes that a change in x-dimension leads to a change in the opposite direction in the y-dimension and vice versa.

it holds that $p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{y})p(\mathbf{y})\,\mathrm{d}\mathbf{y}$ and hence the random variable $\mathbf{y}$ is marginalized. For $p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\boldsymbol{\mu}_\mathbf{y}, \boldsymbol{\Sigma}_\mathbf{y})$ and $p(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\mathbf{x}|\mathbf{A}\mathbf{y} + \mathbf{b}, \boldsymbol{\Sigma}_l)$, the integral $\int \mathcal{N}(\mathbf{x}|\mathbf{A}\mathbf{y} + \mathbf{b}, \boldsymbol{\Sigma}_l)\mathcal{N}(\mathbf{y}|\boldsymbol{\mu}_\mathbf{y}, \boldsymbol{\Sigma}_\mathbf{y})\,\mathrm{d}\mathbf{y}$ is again a MGD. This can be shown by calculating the joint distribution

$$p(\mathbf{x}, \mathbf{y}) = \mathcal{N}\left(\begin{pmatrix}\mathbf{y}\\ \mathbf{x}\end{pmatrix}\middle|\begin{pmatrix}\boldsymbol{\mu}_\mathbf{y}\\ \mathbf{A}\boldsymbol{\mu}_\mathbf{y} + \mathbf{b}\end{pmatrix}, \begin{pmatrix}\boldsymbol{\Sigma}_\mathbf{y} & \boldsymbol{\Sigma}_\mathbf{y}\mathbf{A}^T\\ \mathbf{A}\boldsymbol{\Sigma}_\mathbf{y} & \mathbf{A}\boldsymbol{\Sigma}_\mathbf{y}\mathbf{A}^T + \boldsymbol{\Sigma}_l\end{pmatrix}\right)$$

as shown in [10]. Now the marginal $p(\mathbf{x})$ can be read off the partitioned mean and covariance matrix of the above depiction to be

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{A}\boldsymbol{\mu}_\mathbf{y} + \mathbf{b}, \mathbf{A}\boldsymbol{\Sigma}_\mathbf{y}\mathbf{A}^T + \boldsymbol{\Sigma}_l)$$

as justified in chapter 2 of [2].

2.1.2 Bayes Theorem for Gaussian Distributions

Another common operation in Machine Learning is to combine prior knowledge or assumptions about the learning problem with the sample data $\mathcal{D}$ in order to achieve better or more stable results. A theorem which allows for this incorporation of prior knowledge is called Bayes' rule which is defined as

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{y})p(\mathbf{y})}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathbf{y})p(\mathbf{y})}{\int p(\mathbf{x}|\mathbf{y})p(\mathbf{y})\,\mathrm{d}\mathbf{y}}$$

for two events $\mathbf{x}$ and $\mathbf{y}$ with probability $p(\mathbf{x})$ and $p(\mathbf{y})$. The probability $p(\mathbf{y}|\mathbf{x})$ is commonly referred to as the posterior, while $p(\mathbf{x}|\mathbf{y})$ is called the likelihood, $p(\mathbf{y})$ the prior and $p(\mathbf{x})$ the marginal. The benefit of this rule is that if the prior as well as the likelihood is known, it is possible to calculate the posterior. In the following sections, the prior will encode the prior information about the learning problem so that the

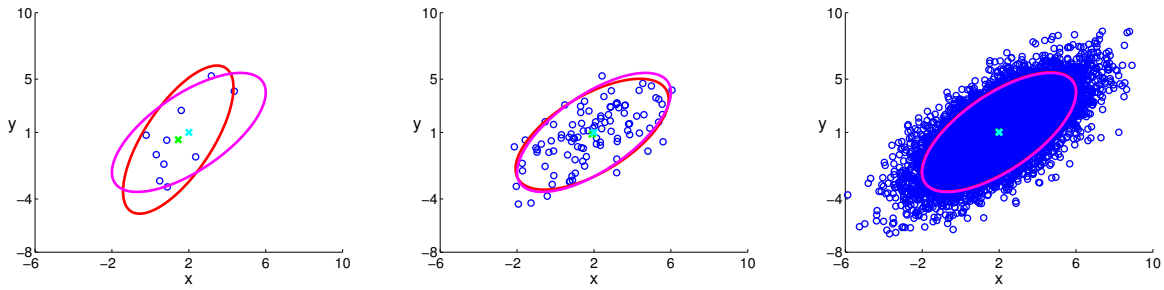

Figure 2.2.: The above plots show Maximum Likelihood Estimates of a two-dimensional Gaussian distribution for growing number of samples (blue circles). The estimated means and covariance matrices are depicted by green crosses and red ellipses. The mean and covariance of the Gaussian distribution from which the data was sampled is depicted by the cyan cross and the magenta ellipse. The left plot shows the estimate for 10, the middle one for 100 and the right one for 10000 samples.

information contained in the sample data will be incorporated through the likelihood. For MGDs there exists a special case of this rule in the form

$$p(\mathbf{y}|\mathbf{x},\mathbf{A},\mathbf{b}) = \frac{\mathcal{N}(\mathbf{x}|\mathbf{A}\mathbf{y}+\mathbf{b},\boldsymbol{\Sigma}_l)\mathcal{N}(\mathbf{y}|\boldsymbol{\mu}_\mathbf{y},\boldsymbol{\Sigma}_\mathbf{y})}{\int \mathcal{N}(\mathbf{x}|\mathbf{A}\mathbf{y}+\mathbf{b},\boldsymbol{\Sigma}_l)\mathcal{N}(\mathbf{y}|\boldsymbol{\mu}_\mathbf{y},\boldsymbol{\Sigma}_\mathbf{y})\,\mathrm{d}\mathbf{y}}$$

where the mean of the likelihood is a linear function of $\mathbf{y}$ specified by $\mathbf{A}$ and $\mathbf{b}$ and the covariances of the likelihood and the prior are independent of each other. For this special case, $p(\mathbf{y}|\mathbf{x},\mathbf{A},\mathbf{b})$ can be expressed in closed form as

$$p(\mathbf{y}|\mathbf{x},\mathbf{A},\mathbf{b}) = \mathcal{N}(\boldsymbol{\Sigma}(\mathbf{A}^T\boldsymbol{\Sigma}_l^{-1}(\mathbf{x}-\mathbf{b})+\boldsymbol{\Sigma}_\mathbf{y}^{-1}\boldsymbol{\mu}_\mathbf{y}),\boldsymbol{\Sigma}), \quad \boldsymbol{\Sigma} = (\boldsymbol{\Sigma}_\mathbf{y}^{-1}+\mathbf{A}^T\boldsymbol{\Sigma}_l^{-1}\mathbf{A})^{-1}.$$

The derivation of this result can be found in chapter 2 of [2]. An alternative formulation of the posterior which will be also used in the following sections is received when first expressing the joint distribution $p(\mathbf{x},\mathbf{y}|\mathbf{A},\mathbf{b}) = \mathcal{N}(\mathbf{x}|\mathbf{A}\mathbf{y}+\mathbf{b},\boldsymbol{\Sigma}_l)\mathcal{N}(\mathbf{y}|\boldsymbol{\mu}_\mathbf{y},\boldsymbol{\Sigma}_\mathbf{y})$ through

$$p(\mathbf{x},\mathbf{y}|\mathbf{A},\mathbf{b}) = \mathcal{N}\left(\begin{pmatrix}\mathbf{y}\\ \mathbf{x}\end{pmatrix}\middle|\begin{pmatrix}\boldsymbol{\mu}_\mathbf{y}\\ \mathbf{A}\boldsymbol{\mu}_\mathbf{y}+\mathbf{b}\end{pmatrix},\begin{pmatrix}\boldsymbol{\Sigma}_\mathbf{y} & \boldsymbol{\Sigma}_\mathbf{y}\mathbf{A}^T\\ \mathbf{A}\boldsymbol{\Sigma}_\mathbf{y} & \mathbf{A}\boldsymbol{\Sigma}_\mathbf{y}\mathbf{A}^T+\boldsymbol{\Sigma}_l\end{pmatrix}\right)$$

as described in [10]. This joint distribution can then be transformed to the posterior

$$p(\mathbf{y}|\mathbf{x},\mathbf{A},\mathbf{b}) = \mathcal{N}(\boldsymbol{\mu}_\mathbf{y}+\boldsymbol{\Sigma}^*(\mathbf{x}-(\mathbf{A}\boldsymbol{\mu}_\mathbf{y}+\mathbf{b})),\boldsymbol{\Sigma}_\mathbf{y}-\boldsymbol{\Sigma}^*\mathbf{A}\boldsymbol{\Sigma}_\mathbf{y}), \quad \boldsymbol{\Sigma}^* = \boldsymbol{\Sigma}_\mathbf{y}\mathbf{A}^T(\mathbf{A}\boldsymbol{\Sigma}_\mathbf{y}\mathbf{A}^T+\boldsymbol{\Sigma}_l)^{-1}$$

according to the results presented in chapter 2 of [2].

2.1.3 Maximum Likelihood Estimation for Gaussian Distributions

The last operation on MGDs which will be relevant for the following sections arises from a field of Machine Learning called Probability Density Estimation (PDE), where given sample data $\mathcal{D} = \{\mathbf{x}_i|i=1,\ldots,N\}$, a parameterized distribution $p(\mathbf{x}|\boldsymbol{\theta})$ is adapted to best explain the sample data. For the sake of completeness, one notes that methods from the field of PDE solve so called Unsupervised Learning problems, where the data set $\mathcal{D}$ does not contain inputs and observed outputs of a function but only observed data. The objective is then to find a structure which best explains this data.

If the distribution is fixed to be a MGD, the parameters which can be adapted are the mean $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$. As the name of this section already suggests, one way to adapt the parameters of a MGD to given sample data is by maximizing the likelihood $L(\mathscr{D}, \boldsymbol{\theta}) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}, \boldsymbol{\Sigma})$ of this data. Consequently, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ can be computed through $\underset{\boldsymbol{\theta}}{\text{argmax}}\, L(\mathscr{D}, \boldsymbol{\theta})$. However, it is easier to calculate $\underset{\boldsymbol{\theta}}{\text{argmax}}\, ln(L(\mathscr{D}, \boldsymbol{\theta}))$ as this removes unnecessary exponential terms in the formula while not altering the result since the ln function is strictly monotonous. The parameters $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ solving this optimization problem are then given by

$$\boldsymbol{\Sigma} = \frac{\sum_{i=1}^{N} (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T}{N}$$
$$\boldsymbol{\mu} = \frac{\sum_{i=1}^{N} \mathbf{x}_i}{N}.$$

The derivation of this result can be reviewed in appendix A.1. Concluding this section about MGDs, figure 2.2 shows an example of Maximum Likelihood Estimation for Gaussian Distributions for the two-dimensional case.

2.2 Regression

Regression refers to a subcategory of Supervised Learning in which the output space of the function f to be learned is continuous, e.g. $O = \mathbb{R}$. Hence, the previously mentioned example of predicting the power consumption of a city can be more precisely referred to as a Regression problem. For the following introduction of various Regression methods, it will be assumed that $I = \mathbb{R}^D$ and $O = \mathbb{R}$, although the presented methods can be applied to other input and continuous output spaces.

2.2.1 Linear Regression

Until now, no further assumptions about the form of the function f were made. In Linear Regression, a linear relationship between inputs and outputs is assumed. Because of that, f can be defined as the vector product between the given input and the parameters to which a constant offset is added, hence

$$f(\mathbf{x}, \boldsymbol{\theta}) = \sum_{i=1}^{D} x_i \cdot \theta_i + \theta_{D+1} = \hat{\mathbf{x}}^T \boldsymbol{\theta}, \quad \hat{\mathbf{x}} = \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix}, \quad \boldsymbol{\theta} \in \mathbb{R}^{D+1}.$$

$\boldsymbol{\theta}$ is commonly referred to as the parameter vector. While this definition allows for the derivation of closed form solutions for Regression problems, fixing the relationship between inputs and outputs to be linear does severely limit the complexity of the problems to which Linear Regression can be applied. Because of that, so called feature functions $\phi : I \mapsto V$ are introduced. The output space of such a feature function is referred to as the Feature Space and will for this thesis be assumed to be $\mathbb{R}^K$. At this point, it is not assumed that the inputs are in a linear relationship with the outputs, but that the transformed inputs have a linear relationship with the outputs. With the resulting definition of f

$$f(\mathbf{x}, \boldsymbol{\theta}) = \sum_{i=1}^{K} \phi(\mathbf{x})_i \cdot \theta_i = \phi(\mathbf{x})^T \boldsymbol{\theta}, \quad \boldsymbol{\theta} \in \mathbb{R}^K$$

it is now possible to express more complex relationships between inputs and outputs while still preserving the ability to derive closed form solutions, since ϕ can be chosen to be non-linear.
Figure 2.3 shows a comparison of different feature functions for a very simple Regression problem, where a method called Least Squares Regression (see section 2.2.1) was used to estimate the parameters.

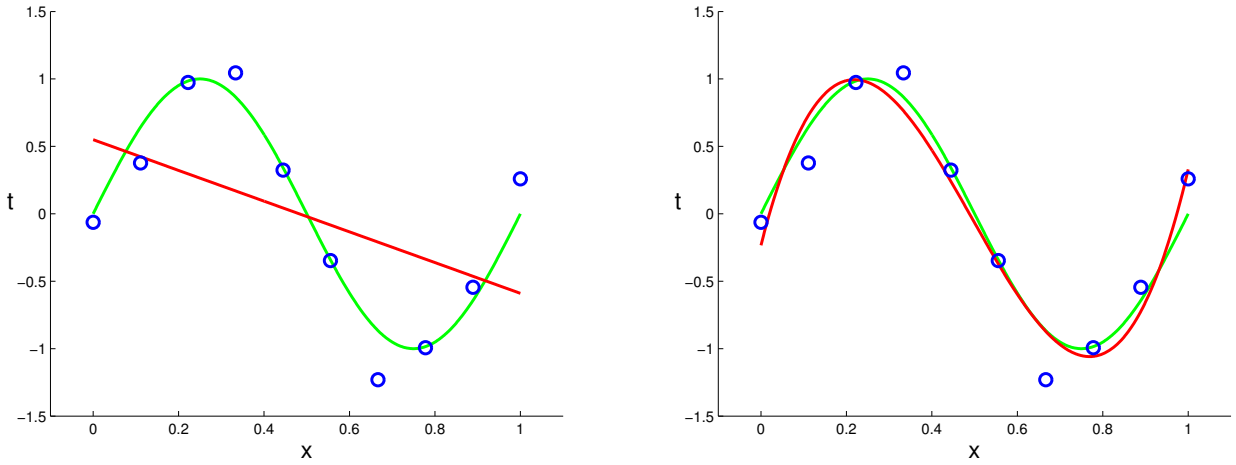

Figure 2.3.: The two plots show a sinusoidal function (green line) from which noisy samples (blue circles) are taken. These samples are used as the sample data for the Linear Regression task. The employed feature function transforms the real input value x into a vector containing monomials of x up to degree M, hence $\phi(x) = [x^0\ x^1\ x^2 \dots\ x^M]^T$. The plot on the left side shows the learned function (red line) for $M = 1$ which corresponds to the assumption of a linear dependency between inputs and outputs. The plot on the right side shows the learned function for $M = 3$. Obviously, for $M = 1$ the resulting function is to simple to express the relationship between inputs and outputs while for $M = 3$ this is possible to a certain extent.

Least Squares / Maximum Likelihood Regression

In order to be able to actually find appropriate parameters $\boldsymbol{\theta}$ for f given sample data $\mathscr{D}$, a definition of J is necessary. An approach to Linear Regression called Least Squares Regression defines J to be the sum of squared differences between predicted outputs for a given sample input and the actually observed outputs

$$J_{LS}(\mathscr{D}, \boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i, \boldsymbol{\theta}))^2 = \sum_{i=1}^{N}(y_i - \phi(\mathbf{x}_i)^T\boldsymbol{\theta})^2, \quad \mathscr{D} = \{(y_i, \mathbf{x}_i)|i = 1, \dots, N\}.$$

For Least Squares Regression, the optimal parameters are found by computing $\underset{\theta}{\operatorname{argmin}}\, J_{LS}(\mathscr{D}, \boldsymbol{\theta})$ yielding

$$\boldsymbol{\theta} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathbf{y}, \quad \boldsymbol{\Phi} = [\phi(\mathbf{x}_1) \dots \phi(\mathbf{x}_N)]^T.$$

The derivations leading to this result can be found in appendix A.2. The same results can be motivated from a probabilistic point of view. For this, the observations y_i are assumed to be subject to random noise $\epsilon \in \mathbb{R}$. Consequently, it is not possible to observe the exact outputs of the function to be learned but only a noisy version of them. If the probability distribution over the random noise is assumed to be a MGD of the form $\mathscr{N}(\epsilon|0, \sigma^2)$, it is possible to define J to be the likelihood of the observed outputs $\mathbf{y}$ given $\boldsymbol{\theta}$ and the transformation of the recorded inputs $\boldsymbol{\Phi}$

$$J_{ML}(\mathscr{D}, \boldsymbol{\theta}) = p(\mathbf{y}|\boldsymbol{\Phi}, \boldsymbol{\theta}) = \mathscr{N}(\mathbf{y}|\boldsymbol{\Phi}\boldsymbol{\theta}, \sigma^2\mathbf{I}).$$

This likelihood can now be maximized in order to find $\boldsymbol{\theta}$. As shown in appendix A.2, the corresponding objective $\underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, J_{ML}(\mathscr{D}, \boldsymbol{\theta})$ can be reformulated to equal $\underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J_{LS}(\mathscr{D}, \boldsymbol{\theta})$.

Maximum-a-Posteriori and Ridge Regression

The reformulation of Least Squares Regression as the maximization of the likelihood of the observed outputs makes the probabilistic view on Regression explicit. The following Regression method called Maximum-a-Posteriori Regression exploits this probabilistic view to include prior information about the parameter vector $\boldsymbol{\theta}$.
Suppose one wants to apply Regression to a set of sample data $\mathcal{D}$ and has engineered a feature function ϕ to transform the input data. Furthermore, the person who engineered the feature function already knows how some of the transformed inputs relate to the produced output of the function, i.e. knows some values of the parameter vector $\boldsymbol{\theta}$. At this point it would be useful to include this information in the Regression method.
By defining a probability distribution $p(\boldsymbol{\theta})$ over $\boldsymbol{\theta}$ it is possible to make use of Bayes' rule to include the prior information as introduced in section 2.1.2. Defining the likelihood function as $p(\mathbf{y}|\boldsymbol{\theta}, \boldsymbol{\Phi}) = \mathcal{N}(\mathbf{y}|\boldsymbol{\Phi}\boldsymbol{\theta}, \sigma^2\mathbf{I})$ and using a MGD as the prior distribution $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_\theta, \boldsymbol{\Sigma}_\theta)$, a closed form expression of the posterior

$$p(\boldsymbol{\theta}|\mathbf{y}, \boldsymbol{\Phi}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\Sigma}(\sigma^{-2}\boldsymbol{\Phi}^T\mathbf{y} + \boldsymbol{\Sigma}_\theta^{-1}\boldsymbol{\mu}_\theta), \boldsymbol{\Sigma}), \quad \boldsymbol{\Sigma} = (\boldsymbol{\Sigma}_\theta^{-1} + \sigma^{-2}\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}$$

can be retrieved. Since a MGD is unimodal with mode $\boldsymbol{\mu}$, choosing the mean as the estimate of the parameter vector $\boldsymbol{\theta}$ leads to maximizing the probability of it. Consequently, the idea behind Maximum-a-Posteriori Regression is to find $\boldsymbol{\theta}$ by calculating $\underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta}|\mathbf{y}, \boldsymbol{\Phi})$, which gives this Regression method its name.
A special case of Maximum-a-Posteriori Regression is the so called Ridge Regression, where the Gaussian prior over $\boldsymbol{\theta}$ is assumed to have zero mean and a spherical covariance, i.e. $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \alpha\mathbf{I})$. This prior leads to the specific posterior distribution

$$p(\boldsymbol{\theta}|\mathbf{y}, \boldsymbol{\Phi}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\Sigma}(\sigma^{-2}\boldsymbol{\Phi}^T\mathbf{y}), \boldsymbol{\Sigma}), \quad \boldsymbol{\Sigma} = (\alpha^{-1}\mathbf{I} + \sigma^{-2}\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}.$$

The intuition of this prior over $\boldsymbol{\theta}$ is that the values in the parameter vector are assumed to be small as long as there is no evidence for the contrary through the data. This assumption can avoid a problem called overfitting. Overfitting occurs, if Regression is done using a complex feature function on comparatively few data. This often leads to the problem, that the model explains the sample data nearly perfectly, but does not approximate the function which generated the data and hence makes wrong predictions for data with which it was not trained. Figure 2.4 illustrates such a problem and how Ridge Regression can be used to circumvent it.

2.2.2 Weighted Linear Regression

As seen in the examples, the success of Linear Regression methods greatly depends on the appropriate choice of the feature function ϕ. If a transformation can be found so that the transformed inputs are in a linear relationship with the outputs, the function f to be fitted is expressive enough to explain the data. An approach which avoids the need of designing such a feature function is referred to as Weighted Linear Regression.
The basic idea behind this approach is that, on an arbitrary small interval, functions can be nearly perfectly approximated by a linear function. Hence the feature function used for Weighted Linear Regression methods commonly takes the form $\phi(\mathbf{x}) = [\mathbf{x}^T 1]^T$ (another common choice is to use cubic polynomials). To be able to represent non-linear functions despite the simple feature function, it is necessary to only include local instead of global information. So if a prediction $f(\mathbf{x}_q)$ for an input $\mathbf{x}_q$ should be made, the relevance of the samples $(y_i, \mathbf{x}_i)$ in $\mathcal{D}$ for the current prediction are expressed by so called weights. A weight w_i for a sample $(y_i, \mathbf{x}_i)$ decreases as some norm of the distance between the input data of the sample and the query data $||\mathbf{x}_q - \mathbf{x}_i||$ increases.

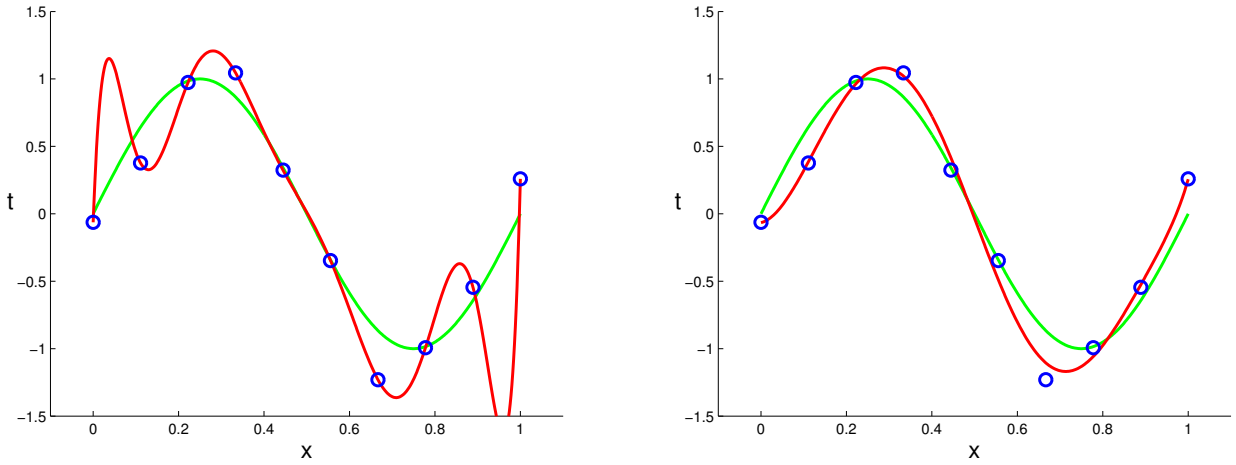

Figure 2.4.: The plots show the same function and samples as in figure 2.3. The left plot shows the learned function using Least Squares Regression with the same feature function used in figure 2.3 and $M = 9$. The right plot shows the learned function for $M = 9$ using Ridge Regression and $ln(\lambda) = ln(\frac{\sigma^2}{\alpha}) = -18$. Obviously, training the complex model on only 10 samples of the function leads to overfitting which can be circumvented by employing Ridge Regression.

Commonly a weight is computed by applying a non-linear transformation $K : \mathbb{R} \mapsto \mathbb{R}$, such as a so called Gaussian Kernel $K(d) = exp(-d^2)$, to the computed distance between sample and query data. Often a parameter h is introduced to scale the distance measure before applying the non-linear transformation. This parameter can then be adapted to influence how much data should be taken into account for the current prediction. All in all, this leads to $w_i = K\left(\frac{||\mathbf{x}_q - \mathbf{x}_i||}{h}\right)$.

Weighted Least Squares / Weighted Maximum Likelihood Regression

As the name of this Regression method suggests, a similar cost function as in Least Squares Regression is used. However, this time it is extended with the calculated weights. Since the weights w_i are calculated based on the query point $\mathbf{x}_q$, the cost function

$$J_{WLS}(\mathscr{D}, \boldsymbol{\theta}, \mathbf{x}_q) = \sum_{i=1}^{N} w_i (y_i - \phi(\mathbf{x}_i)^T \boldsymbol{\theta})^2, \quad w_i = K\left(\frac{||\mathbf{x}_q - \mathbf{x}_i||}{h}\right)$$

is extended to depend also on this query point. Solving $\underset{\boldsymbol{\theta}}{\operatorname{argmin}} J_{WLS}(\mathscr{D}, \boldsymbol{\theta}, \mathbf{x}_q)$ leads to

$$\boldsymbol{\theta} = (\boldsymbol{\Phi}^T \mathbf{W} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \mathbf{W} \mathbf{y}, \quad \mathbf{W} = \text{diag}(w_1, \ldots, w_N)$$

as shown in appendix A.3. An example for Weighted Least Squares Regression is depicted in figure 2.5 where the idea of linear approximations and the influence of the bandwidth h on them is shown.
At this point it is once more useful to reformulate Weighted Least Squares Regression as the maximization of

$$J_{WML}(\mathscr{D}, \boldsymbol{\theta}, \mathbf{x}_q) = ln(\mathscr{N}(\mathbf{y} | \boldsymbol{\Phi}\boldsymbol{\theta}, \mathbf{W}^{-1}))$$

as shown in appendix A.3. Note that the covariance matrix of the defined MGD is set to the inverse of $\mathbf{W}$, hence for a sample $(y_i, \mathbf{x}_i)$ with small weight w_i, the corresponding entry in the diagonal of the covariance matrix will be big. Intuitively this means that the variance in this dimension of the Gaussian distribution is high and hence a deviation of $\phi(\mathbf{x}_i)^T \boldsymbol{\theta}$ from y_i is more likely than for a sample with a big weight. Obviously this intuition is the same as the one behind the Least Squares cost function.

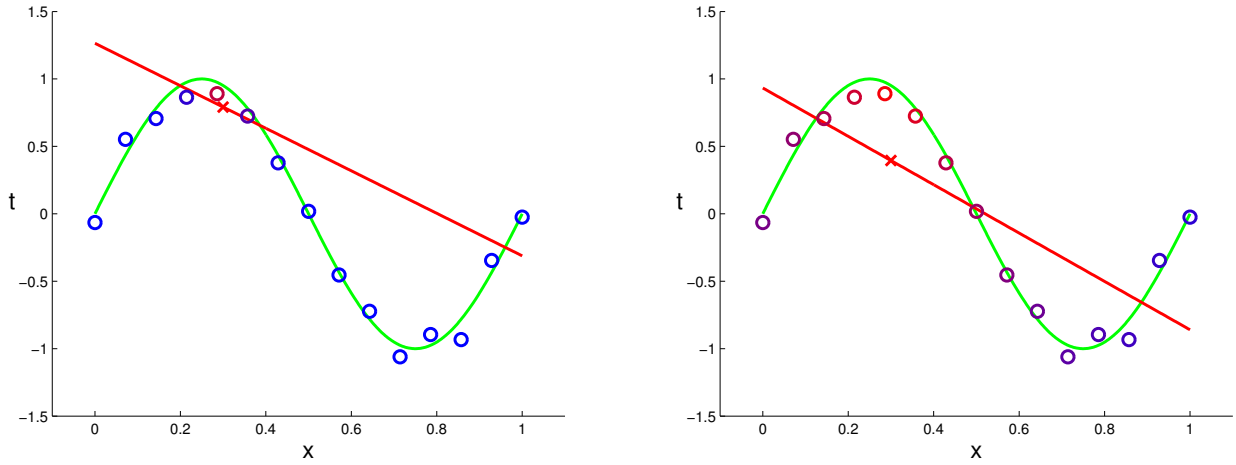

Figure 2.5.: The two plots show the function to learn (green curve), the noisy samples taken from it (circles), the prediction for the query point $x_q = 0.3$ (red cross) as a result of Weighted Linear Regression and the corresponding linear approximation of the function (red line) at that point. The weights used for Regression were calculated by $w_i = exp\left(-\frac{||x_q - x_i||^2}{h}\right)$. The color of a circle represents the size of the weight associated with the sample, where red stands for a large weight and blue for a small weight. In the left plot h is set to 0.05 while in the right it is set to 0.4.

Weighted Maximum-a-Posteriori Regression

Just as before, the goal of Weighted Maximum-a-Posteriori Regression is to include prior information about the parameter vector $\boldsymbol{\theta}$. For computational tractability, the prior information is again expressed through a MGD $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_\theta, \boldsymbol{\Sigma}_\theta)$. Using the same results from section 2.1.2 leads to

$$p(\boldsymbol{\theta}|\mathbf{y}, \boldsymbol{\Phi}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\Sigma}(\boldsymbol{\Phi}^T\mathbf{W}\mathbf{y} + \boldsymbol{\Sigma}_\theta^{-1}\boldsymbol{\mu}_\theta), \boldsymbol{\Sigma}), \quad \boldsymbol{\Sigma} = (\boldsymbol{\Sigma}_\theta^{-1} + \boldsymbol{\Phi}^T\mathbf{W}\boldsymbol{\Phi})^{-1}$$

as the closed form expression of the posterior distribution over $\boldsymbol{\theta}$.

2.2.3 Gaussian Process Regression

Besides Weighted Linear Regression, so called Gaussian Process Regression is another Regression method that avoids the need of designing feature functions. The name of the method comes from a statistical model called Gaussian Process. Gaussian Processes can be seen as a distribution over functions $f : I \mapsto O$. Just as before, it is assumed that $I = \mathbb{R}^D$, while $O = \mathbb{R}$. According to [11], such a Gaussian Process can be defined by a mean and covariance function

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$$
$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}) - m(\mathbf{x}))^T]$$

where $\mathbb{E}[\cdot]$ is the expectation operator. For any $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subseteq I$ and $\mathbf{y} = [y_1 \ \ldots \ y_N]^T \in O^N$, the joint distribution

$$p(\mathbf{y}|\mathbf{X}) = \mathcal{N}\left(\mathbf{y} \middle| \begin{pmatrix} m(\mathbf{x}_1) \\ \vdots \\ m(\mathbf{x}_N) \end{pmatrix}, \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}\right)$$

is now a MGD. Assuming f to be of linear form $f(\mathbf{x}) = \phi(\mathbf{x})^T\boldsymbol{\theta}$ in feature space for some feature function $\phi(\cdot)$ and using $\mathcal{N}(\boldsymbol{\theta}|\mathbf{0},\mathbf{I})$ as the prior over $\boldsymbol{\theta}$, it is possible to express m and k in closed form

$$m(\mathbf{x}) = \mathbb{E}[\phi(\mathbf{x})^T\boldsymbol{\theta}] = \phi(\mathbf{x})^T\mathbb{E}[\boldsymbol{\theta}] = 0$$
$$k(\mathbf{x},\mathbf{x}') = \mathbb{E}[\phi(\mathbf{x})^T\boldsymbol{\theta}\boldsymbol{\theta}^T\phi(\mathbf{x}')] = \phi(\mathbf{x})^T\mathbb{E}[\boldsymbol{\theta}\boldsymbol{\theta}^T]\phi(\mathbf{x}') = \phi(\mathbf{x})^T\phi(\mathbf{x}').$$

Naturally, the prior over $\boldsymbol{\theta}$ can be chosen differently, but for the purpose of this thesis it is assumed to be of the mentioned form. Until now, the feature function needs to be explicitly specified. However, it is possible to replace the scalar product $\phi(\mathbf{x})^T\phi(\mathbf{x}')$ by a so called kernel function $\kappa : I \times I \mapsto \mathbb{R}$. Such a kernel function needs to fulfill certain properties to be an appropriate replacement for the scalar product but intuitively speaking, it needs to measure the similarity between $\mathbf{x}$ and $\mathbf{x}'$ in some feature space. A common choice for κ is

$$\kappa(\mathbf{x},\mathbf{x}') = \omega_1 exp\left((\mathbf{x}-\mathbf{x}')^T\mathbf{Q}(\mathbf{x}-\mathbf{x}')\right), \quad \mathbf{Q} = diag(\omega_2,\ldots,\omega_{D+1})$$

which is referred to as the squared exponential kernel. Now the sample data $\mathscr{D}$ can be interpreted as samples from a Gaussian Process leading to

$$p(\mathbf{y}|\mathbf{X}) = \mathcal{N}(\mathbf{y}|\mathbf{0},\mathbf{K})$$

with $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i,\mathbf{x}_j)$. Assuming Gaussian Noise $\mathcal{N}(\epsilon|0,\omega_{D+2}^{-1})$ on the samples y_i, the distribution over the sample outputs $\mathbf{y}$ changes to

$$p(\mathbf{y}|\mathbf{X}) = \mathcal{N}(\mathbf{y}|\mathbf{0},\mathbf{C}) = \int \mathcal{N}(\mathbf{y}|\mathbf{y}',\omega_{D+2}\mathbf{I})\mathcal{N}(\mathbf{y}'|\mathbf{0},\mathbf{K})\,d\mathbf{y}'$$

where $\mathbf{C} = \mathbf{K} + \omega_{D+2}\mathbf{I}$ (see section 2.1.1 for the derivation of the closed form solution of the integral). Given a new input $\mathbf{x}^*$, the probability $p(y^*|\mathbf{x}^*,\mathbf{y},\mathbf{X})$ for $y^* = f(\mathbf{x}^*)$ can be expressed by

$$\mathcal{N}(y^*|\mathbf{k}^T\mathbf{C}^{-1}\mathbf{y}, c - \mathbf{k}^T\mathbf{C}^{-1}\mathbf{k})$$

where $\mathbf{k} = [\kappa(\mathbf{x_1},\mathbf{x}^*)\ \ldots\ \kappa(\mathbf{x_N},\mathbf{x}^*)]^T$ and $c = \kappa(\mathbf{x}^*,\mathbf{x}^*) + \omega_{D+2}$. This distribution is also referred to as the predictive distribution. Details on the derivation of this result can be found in chapter six of [2]. Note that this distribution does not only encode the prediction for y^* (which is given by the mean), but also the uncertainty in the prediction (given by the covariance matrix).
Finally, a relevant task when using Gaussian Process Regression is the optimization of so called hyper-parameters, which are in this case $\boldsymbol{\omega} = [\omega_1\ \ldots\ \omega_{D+2}]^T$. There exist different methods for doing this. A method used in this thesis is based on the maximization of the log likelihood $ln(\mathcal{N}(\mathbf{y}|\mathbf{0},\mathbf{K}))$ of the sample data. In general, the maximization of this likelihood is a non-linear optimization problem.
Concluding this section, figure 2.6 shows Gaussian Process Regression applied to the Regression problem from section 2.2.1.

2.3 ProMPs

As mentioned in chapter 1, ProMPs are a method which allow for the compact representation of not only movements but distributions over movements, hence multiple alternatives of a movement. For this thesis, movements of a robot will be represented by the recorded positions, velocities and/or accelerations of the joints which make up the robot.
To pose an example for this representation of movements, suppose a movement of a robotic arm consisting of $D = 7$ joints was recorded over 3 seconds with a recording frequency of 500 Hz. This leads

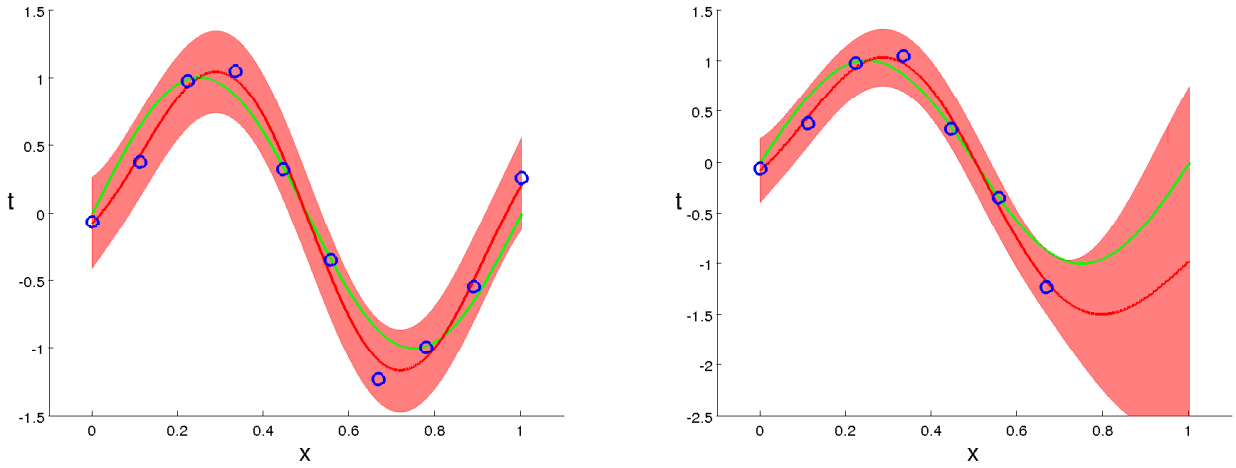

Figure 2.6.: The two plots show the same Regression problem as in figure 2.3 to which Gaussian Process Regression is applied. The red line is the mean of the predictive distribution while the shaded area is two times its standard deviation. In the left plot, the complete set of samples (blue circles) was used while some of them were missing in the right plot. The resulting information loss reflects in a higher uncertainty in the corresponding input area.

to a total of $T = 1500$ samples representing the movement. The timestamps of those samples are denoted as $t_1, \dots, t_T$ with $t_1 = 0$ and $t_T = 3$. The position of joint i at time t_j is denoted with q_{i_j}, where $i \in [1, \dots, 7]$ and $j \in [1, \dots, 1500]$. Similarly, the velocity and acceleration of the same joint at the same time is denoted with $\dot{q}_{i_j}$ and $\ddot{q}_{i_j}$.
In the following sections, the joint positions during a movement will be of particular interest. The recorded positions of joint i during an executed movement will be referred to as a position or sample trajectory $\boldsymbol{\tau}_{pos_i} = [q_{i_1} \ \dots \ q_{i_T}]^T$. Since ProMPs work on multiple of such sample trajectories, a superscript index will be added to distinguish between them. Consequently, the j-th sample trajectory of a set of such will be referred to as $\boldsymbol{\tau}^j_{pos_i} = [q^j_{i_1} \ \dots \ q^j_{i_T}]^T$.

2.3.1 Trajectory Representation

ProMPs work on multiple samples of a movement $\boldsymbol{\tau}^j_{pos_i}$ with $j = 1, \dots, N$. The first step for deriving a distribution over the movement samples is to find a compact representation of every single movement. This compact representation will be the result of a Linear Regression. The features for this Regression problem are so called Radial Basis Functions, such as Gaussian Basis Functions

$$\phi^G_l(t) = exp\left(-\frac{(t - c_l)^2}{2h}\right).$$

Those Gaussian Basis Functions have a center c_l, where their activation is the highest, and a bandwidth h. It is common to spread the centers of those basis functions evenly spaced over the duration of the movement. Note that there are other Basis Functions which e.g. enable the representation of rhythmic movements. However, for the purpose of this thesis only Gaussian Basis Functions will be of interest as the focus is on stroke-based movements. Depending on the length and desired resolution, the number M of basis functions can be chosen to form the feature vector

$$\boldsymbol{\phi}(t) = \left[\frac{\phi^G_1(t)}{\sum_{l=1}^M \phi^G_l(t)} \ \cdots \ \frac{\phi^G_M(t)}{\sum_{l=1}^M \phi^G_l(t)}\right]^T$$

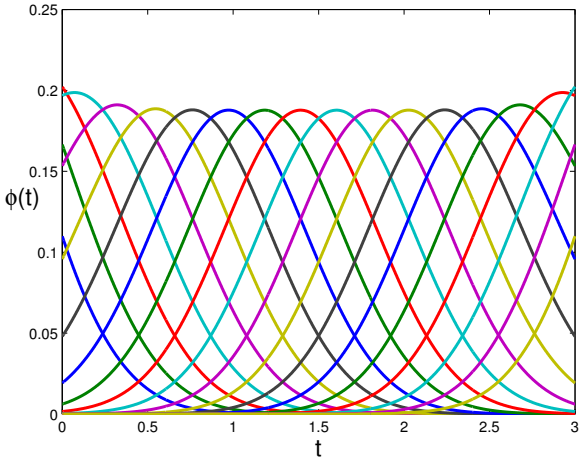

Figure 2.7.: This plot shows the activations of the features in the feature vector $\boldsymbol{\phi}(t)$ for 20 Gaussian Basis Functions with bandwidth $h = 0.2$ and centers $c_i \approx -0.5 + 0.2105 \cdot i$ in the time interval $[0, 3]$. Each line corresponds to the activation of one Gaussian Basis Function in the feature vector over time.

for a given timestamp t. Now the feature matrix for all the time stamps $t_1, \ldots, t_T$ of a trajectory is expressed by

$$\boldsymbol{\Phi} = [\boldsymbol{\phi}(t_1) \ \ldots \ \boldsymbol{\phi}(t_T)]^T.$$

Figure 2.7 illustrates the activations of the single features in $\boldsymbol{\Phi}$ over time. The probability for observing a sample trajectory is then defined as

$$p(\boldsymbol{\tau}^j_{pos_i}) = \mathcal{N}(\boldsymbol{\tau}^j_{pos_i} | \boldsymbol{\Phi}\mathbf{w}_j, \boldsymbol{\Sigma}_\tau)$$

where $\boldsymbol{\Sigma}_\tau \in \mathbb{R}^{T \times T}$ encodes the assumed observation noise. For this thesis, it is supposed that $\boldsymbol{\Sigma}_\tau = \sigma^2 \mathbf{I}$. The parameter vectors $\mathbf{w}_j \in \mathbb{R}^M$ representing the sample trajectories can then be computed using e.g. Maximum Likelihood Regression.
In order to capture the variance between the sample trajectories, a Gaussian distribution $\mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_\mathbf{w}, \boldsymbol{\Sigma}_\mathbf{w})$ is placed over the parameter vectors $\mathbf{w}_j$. The parameters $\boldsymbol{\mu}_\mathbf{w}$ and $\boldsymbol{\Sigma}_\mathbf{w}$ of the distribution can be estimated by Maximum Likelihood Estimation as presented in section 2.1.3. With this distribution, $\mathbf{w}$ can be marginalized as shown in section 2.1.1 to receive the final distribution over the trajectory

$$p(\boldsymbol{\tau}) = \mathcal{N}(\boldsymbol{\tau} | \boldsymbol{\Phi}\boldsymbol{\mu}_\mathbf{w}, \boldsymbol{\Sigma}_\tau + \boldsymbol{\Phi}\boldsymbol{\Sigma}_\mathbf{w}\boldsymbol{\Phi}^T).$$

In the following, this distribution will be referred to as the learned trajectory distribution. Figure 2.8 shows an example of a ProMP being trained for a movement of a single joint. At this point it is worth noticing that ProMPs can also be trained for multiple joints. For that, the complete position trajectory $\boldsymbol{\tau}^j_{pos}$ and the necessary feature matrix $\boldsymbol{\Psi}$ are defined as

$$\boldsymbol{\tau}^j_{pos} = \begin{pmatrix} \boldsymbol{\tau}^j_{pos_1} \\ \vdots \\ \boldsymbol{\tau}^j_{pos_D} \end{pmatrix} \qquad \boldsymbol{\Psi} = \begin{pmatrix} \boldsymbol{\Phi} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Phi} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{\Phi} \end{pmatrix}$$

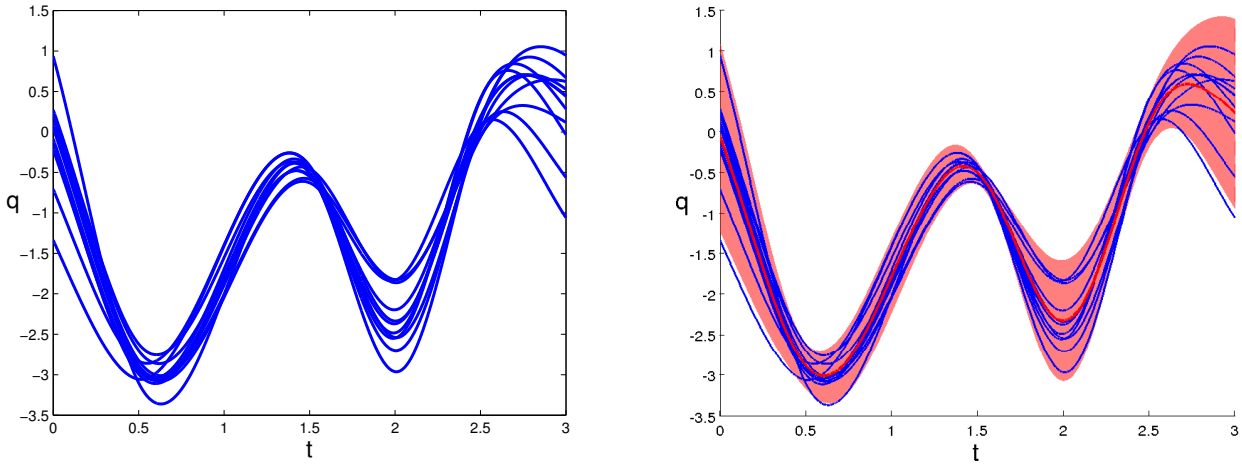

Figure 2.8.: The left plot shows 10 sample trajectories (blue lines) of a movement of a single joint for a duration of 3 seconds. Using the feature matrix arising from the basis functions illustrated in figure 2.7, the Gaussian distribution over the trajectories is learned. This distribution is shown in the right plot, where the red line represents its mean and the shaded area in red corresponds to two times its standard deviation.

Now the probability for observing a sample trajectory is defined as $\mathcal{N}(\boldsymbol{\tau}_{pos}^{j}|\boldsymbol{\Psi}\mathbf{w}_j, \boldsymbol{\Sigma}_\tau)$ and with that the same methods for estimating $\boldsymbol{\mu}_\mathbf{w}$ and $\boldsymbol{\Sigma}_\mathbf{w}$ can be used. However, the resulting distribution now also encodes the coupling between the joints, which would not be the case if a ProMP would be trained for each joint independently. Furthermore, note that $\boldsymbol{\Sigma}_\tau$ and $\mathbf{w}_j$ need to be extended so that $\boldsymbol{\Sigma}_\tau \in \mathbb{R}^{(D\cdot T)\times(D\cdot T)}$ and $\mathbf{w}_j \in \mathbb{R}^{D\cdot M}$.

2.3.2 Model-Based Controller for ProMPs

Besides various operations on movement distributions, the ProMP framework allows for the derivation of a model-based control law which creates trajectories represented by the distribution. This control law is based on a linear dynamics model

$$\mathbf{y}_{t+dt} = (\mathbf{I} + \mathbf{A}_t dt)\mathbf{y}_t + \mathbf{B}_t dt\mathbf{u} + \mathbf{c}_t dt \tag{2.1}$$

where $\mathbf{y}_t = [q_{1_t}\ \dot{q}_{1_t}\ \dots\ q_{D_t}\ \dot{q}_{D_t}]^T$ is the state of the system which is defined to be the joint positions and velocities at time point t. The vector $\mathbf{u}$ contains the motor signals applied to the system. $\mathbf{A}_t$, $\mathbf{B}_t$ and $\mathbf{c}_t$ will be referred to as the system matrices or the linearized dynamics. The control law consists of a so called feedback and feedforward term $\mathbf{K}_t$ and $\mathbf{k}_t$, from which the motor signals are computed by

$$\mathbf{u} = \mathbf{K}_t\mathbf{y}_t + \mathbf{k}_t.$$

Substituting this control law into the linear dynamics model allows to reformulate the model as a linear transformation of the system state $\mathbf{y}_t$

$$\mathbf{y}_{t+dt} = (\mathbf{I} + \mathbf{A}_t dt)\mathbf{y}_t + \mathbf{B}_t dt\mathbf{u} + \mathbf{c}_t dt = \mathbf{F}_t\mathbf{y}_t + \mathbf{f}_t$$

where $\mathbf{F}_t = \mathbf{I} + (\mathbf{A}_t + \mathbf{B}_t\mathbf{K}_t)dt$ and $\mathbf{f}_t = (\mathbf{B}_t\mathbf{k}_t + \mathbf{c}_t)dt$. Although ProMPs are trained only on position trajectories, it is still possible to express a distribution over the system state by using the derivative of the feature function $\boldsymbol{\phi}(t)$

$$\mathcal{N}(\mathbf{y}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t), \quad \boldsymbol{\mu}_t = \boldsymbol{\Psi}_t\boldsymbol{\mu}_\mathbf{w}, \quad \boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_\mathbf{y} + \boldsymbol{\Psi}_t\boldsymbol{\Sigma}_\mathbf{w}\boldsymbol{\Psi}_t^T, \quad \boldsymbol{\Psi}_t = \begin{pmatrix} \boldsymbol{\phi}'(t)^T & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\phi}'(t)^T & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{\phi}'(t)^T \end{pmatrix}$$

where $\boldsymbol{\phi}'(t) = [\boldsymbol{\phi}(t)\ \dot{\boldsymbol{\phi}}(t)]$ and $\boldsymbol{\Sigma}_\mathbf{y} = \sigma^2\mathbf{I}$ is again observation noise. The influence of the control law employed at time step t on the system state can now be expressed by

$$p(\mathbf{y}_{t+dt}) = \int \mathcal{N}(\mathbf{y}_{t+dt}|\mathbf{F}_t\mathbf{y}_t + \mathbf{f}_t, \boldsymbol{\Sigma}_s dt)\mathcal{N}(\mathbf{y}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)d\mathbf{y}_t$$

where the covariance matrix $\boldsymbol{\Sigma}_s$ represents the system noise and arises from the assumption of Gaussian noise on the motor signals $\mathbf{u} = \mathbf{K}_t\mathbf{y}_t + \mathbf{k}_t + \boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \frac{\boldsymbol{\Sigma}_\mathbf{u}}{dt})$, leading to $\boldsymbol{\Sigma}_s = \mathbf{B}_t\boldsymbol{\Sigma}_\mathbf{u}\mathbf{B}_t^T dt$. Using the results from section 2.1.1, the distribution over the next system state can be defined as

$$p(\mathbf{y}_{t+dt}) = \mathcal{N}(\mathbf{y}_{t+dt}|\mathbf{F}_t\boldsymbol{\mu}_t + \mathbf{f}_t, \mathbf{F}_t\boldsymbol{\Sigma}_t\mathbf{F}_t^T + \boldsymbol{\Sigma}_s dt).$$

However, the distribution over this next state can also be defined using $\boldsymbol{\mu}_{t+dt} = \boldsymbol{\Psi}_{t+dt}\boldsymbol{\mu}_\mathbf{w}$ and $\boldsymbol{\Sigma}_{t+dt} = \boldsymbol{\Psi}_{t+dt}\boldsymbol{\Sigma}_\mathbf{w}\boldsymbol{\Psi}_{t+dt}^T$. Consequently, it is possible to define two constraints

$$\begin{aligned} \boldsymbol{\mu}_{t+dt} &= \mathbf{F}_t\boldsymbol{\mu}_t + \mathbf{f}_t \\ \boldsymbol{\Sigma}_{t+dt} &= \mathbf{F}_t\boldsymbol{\Sigma}_t\mathbf{F}_t^T + \boldsymbol{\Sigma}_s dt \end{aligned}$$

which allow for the derivation of $\mathbf{K}_t$ and $\mathbf{k}_t$. For brevity of this section, the derivations are skipped and only the results are presented. However, they can be reviewed in [9].

$$\mathbf{K}_t = \mathbf{B}_t^\dagger(\dot{\boldsymbol{\Psi}}_t\boldsymbol{\Sigma}_\mathbf{w}\boldsymbol{\Psi}_t^T - \mathbf{A}_t\boldsymbol{\Sigma}_t - \frac{\boldsymbol{\Sigma}_s}{2})\boldsymbol{\Sigma}_t^{-1} \tag{2.2}$$

$$\mathbf{k}_t = \mathbf{B}_t^\dagger(\dot{\boldsymbol{\Psi}}_t\boldsymbol{\mu}_\mathbf{w} - (\mathbf{A}_t + \mathbf{B}_t\mathbf{K}_t)\boldsymbol{\Psi}_t\boldsymbol{\mu}_\mathbf{w} - \mathbf{c}_t) \tag{2.3}$$

In the above and following equations, $\mathbf{B}_t^\dagger$ refers to the pseudo-inverse of $\mathbf{B}_t$. At the beginning of the controller derivation, a Gaussian noise with zero mean on the motor signals was assumed. However, the covariance matrix $\boldsymbol{\Sigma}_\mathbf{u}$ was assumed to be given. This covariance matrix can also be computed by matching the covariance matrices of the joint distributions $p(\mathbf{y}_t, \mathbf{y}_{t+dt})$ received by the ProMP representation $\mathcal{N}(\mathbf{y}_{t+dt}|\boldsymbol{\mu}_{t+dt}, \boldsymbol{\Sigma}_{t+dt})\mathcal{N}(\mathbf{y}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ and the system dynamics $\mathcal{N}(\mathbf{y}_{t+dt}|\mathbf{F}_t\mathbf{y}_t + \mathbf{f}_t, \boldsymbol{\Sigma}_s dt)\mathcal{N}(\mathbf{y}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$. For the derivation, it is again referred to [9] and only the result

$$\boldsymbol{\Sigma}_\mathbf{u} = \mathbf{B}_t^\dagger\boldsymbol{\Sigma}_s\mathbf{B}_t^{\dagger T}, \quad \boldsymbol{\Sigma}_s dt = \boldsymbol{\Sigma}_{t+dt} - \mathbf{C}_t^T\boldsymbol{\Sigma}_t^{-1}\mathbf{C}_t, \quad \mathbf{C}_t = \boldsymbol{\Psi}_t\boldsymbol{\Sigma}_\mathbf{w}\boldsymbol{\Psi}_{t+dt}^T$$

is presented. Finally, using this model-based controller to reproduce the learned trajectory distribution will be referred to as the execution of a ProMP.

3 Model Learning for ProMPs

This thesis evaluates methods for learning linear dynamics for model-based control with ProMPs. Specifically, the system matrices $\mathbf{A}_t$, $\mathbf{B}_t$ and $\mathbf{c_t}$ employed in the derivation of the feeback and feedforward terms of the controller (see section 2.3.2) need to be learned. For this learning task, we assume that we can sample N trajectories that contain positions and velocities τ^j_{learn} as well as the motor signals u^j_{learn} used to create those trajectories, i.e.

$$\tau^j_{learn} = [\mathbf{y}^j_1 \ \dots \ \mathbf{y}^j_T]^T, \quad u^j_{learn} = [\mathbf{u}^j_1 \ \dots \mathbf{u}^j_{T-1}]^T$$

where $j \in [1, \dots, N]$. During the execution of the model-based controller, new movement trajectories and corresponding motor signals can be observed, which can be used to improve the prediction of the system matrices.

3.1 Learning Architecture

In this section we present relevant properties of the evaluated Regression methods such as sample data transformation, parameter priors or the distance metric for Weighted Linear Regression. Furthermore, we motivate approaches which compensate for errors in the learned system matrices and examine the relevance of prediction errors in the single system matrices.

3.1.1 Feature function and Sample Data

For Linear Regression methods, it is possible to define an input-output relationship so that the learned parameter vector directly corresponds to the system matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{c}$, i.e. by rewriting equation 2.1 we obtain

$$\begin{aligned}
\mathbf{y}^j_{t+dt} &= (\mathbf{I} + \mathbf{A}dt)\mathbf{y}^j_t + \mathbf{B}dt\mathbf{u}^j_t + \mathbf{c}dt \\
\mathbf{y}^j_{t+dt} - \mathbf{y}^j_t &= (\mathbf{A}\mathbf{y}^j_t + \mathbf{B}\mathbf{u}^j_t + \mathbf{c})dt \\
\frac{\mathbf{y}^j_{t+dt} - \mathbf{y}^j_t}{dt} &= \mathbf{A}\mathbf{y}^j_t + \mathbf{B}\mathbf{u}^j_t + \mathbf{c} \\
\frac{\mathbf{y}^j_{t+dt} - \mathbf{y}^j_t}{dt} &= \begin{pmatrix} \mathbf{A} & \mathbf{B} & \mathbf{c} \end{pmatrix} \begin{pmatrix} \mathbf{y}^j_t \\ \mathbf{u}^j_t \\ 1 \end{pmatrix} \\
\dot{\mathbf{y}}^j_t &= \boldsymbol{\theta}\mathbf{x}^j_t. \qquad (3.1)
\end{aligned}$$

In order to obtain the derived correspondence between parameter vector and system matrices, the set of sample data needs to be defined as $\mathscr{D} = \{(\mathbf{x}_t^j, \dot{\mathbf{y}}_t^j) | t \in [1, \ldots, T-1], j \in [1, \ldots, N]\}$. We apply 3.1 to every dimension of $\dot{\mathbf{y}}_t^j$ individually. Hence, the matrix $\boldsymbol{\theta}$ consists of $2 \cdot D$ parameter vectors $\boldsymbol{\theta}_i$

$$\dot{\mathbf{y}}_t^j = \begin{pmatrix} \dot{q}_{1_t}^j \\ \ddot{q}_{1_t}^j \\ \vdots \\ \dot{q}_{D_t}^j \\ \ddot{q}_{D_t}^j \end{pmatrix} = \begin{pmatrix} \boldsymbol{\theta}_1 \\ \vdots \\ \boldsymbol{\theta}_{2D} \end{pmatrix} \mathbf{x}_t^j = \boldsymbol{\theta} \mathbf{x}_t^j$$

where D is the number of joints.

3.1.2 Parameter Prior

Wrong predictions of the linear dynamics model can result in motor signals which will most likely not be able to reproduce the desired trajectory distribution or even damage the robot or its environment. Therefore, we include prior information about the dynamics model to improve the estimation accuracy of the learning methods.
For clearness, the prior is motivated on a linear dynamics model for a robot with only one joint. However, the extension to multiple joints is straightforward. For a robot with only one joint, the linear dynamics model can be expressed by

$$\dot{\mathbf{y}} = \begin{pmatrix} \dot{q} \\ \ddot{q} \end{pmatrix} = \begin{pmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{pmatrix} \begin{pmatrix} q \\ \dot{q} \end{pmatrix} + \begin{pmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \end{pmatrix} u + \begin{pmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{pmatrix} = \mathbf{A}\mathbf{y} + \mathbf{B}\mathbf{u} + \mathbf{c}.$$

Setting $\boldsymbol{\theta}_1 = [\mathbf{A}_{1,1}\ \mathbf{A}_{1,2}\ \mathbf{B}_1\ \mathbf{c}_1] = [0\ 1\ 0\ 0]$ will accomplish $\dot{\mathbf{y}}_1 = \dot{q} = \boldsymbol{\theta}_1 \mathbf{x} = \boldsymbol{\theta}_1 [q\ \dot{q}\ u\ 1]^T$. Consequently, it is not required to actually learn $\boldsymbol{\theta}_1$ as its entries are known from the form of the dynamical system at hand. For a robot with multiple joints, such a result can be obtained for every parameter vector which is used to predict $\dot{q}_i$ with $i \in [1, \ldots, D]$. Nevertheless, in the experiments the prior was used to check whether the evaluated Regression methods tend to explain the relationship between $\dot{q}_i$ and $\mathbf{x}$ differently. For that the covariance matrix of the prior distribution over the parameters was set to a diagonal matrix with small diagonal entries, such as 10^{-5} and the resulting parameter posterior was examined for significant deviations from the prior.

3.1.3 Distance Metric for Weighted Linear Regression

For Weighted Linear Regression, a poorly chosen distance metric can limit the performance of the resulting Regression method. Consequently, there exist many different suggestions for such metrics and ways how to optimize their parameters, such as the bandwidth h [12] [13] [14]. For control, the computation of the weights should ideally be fast but still adapt to the density of samples which are similar to the current query.
The metric used in this thesis combines approaches presented in [12]. For the calculation of the weights, a so called tricube kernel is used as the nonlinear transformation applied to the euclidean distance between the query point and the sample inputs. Note that the euclidean distance is computed on the system state $\mathbf{y}_t^j$, discarding the corresponding motor signal $\mathbf{u}_t^j$, as during the execution of a ProMP, the motor signals can only be computed after the linear dynamics model is known. Furthermore, this

expresses the assumption that the actions (which correspond to the motor signals), do not affect the system matrices **A**, **B** and **c**. The scaling parameter h is adjusted using a nearest neighbor approach, setting it to the distance to the $(K+1)$-th nearest neighbor. All in all the distance metric can be defined as

$$w_t^j = d(\mathbf{y}_q, \mathbf{x}_t^j) = K\left(\frac{\|\mathbf{y}_q - \mathbf{y}_t^j\|_2}{h_{K-NN}}\right)$$
$$K(d) = \begin{cases}(1-|d|^3)^3, & |d|<1\\ 0, & |d|>=1\end{cases}$$
$$h_{K-NN} = \underset{\mathbf{x}\in\mathscr{D}}{(\mathrm{K}+1)\text{-min}}(\|\mathbf{y}_q, \mathbf{y}\|_2)$$

where $K(d)$ is the mentioned tricube kernel and the operator $(\mathrm{K}+1)$-min specifies the $(K+1)$-th smallest distance between the query point and the inputs from the sample data. An implication of this distance metric is that only the K nearest neighbors to the query point are taken into account for the prediction as for all other inputs in the set of sample data, the distance is at least as big as the distance to the $(K+1)$-th nearest neighbor, i.e. the scaling parameter h_{K-NN}. Hence, the final weight for those inputs will be zero. This allows for the use of efficient nearest neighbor search algorithms, using e.g. variations of KD-Trees [15].

3.1.4 Backup PD Controller and Clipping of Motor Signals

Ideally, the learning methods would only need the sample trajectories used to learn the trajectory distribution in order to correctly predict the system matrices. However, for complex dynamical systems, the collected data often does not suffice to accomplish this goal and hence errors in the system matrix predictions lead to motor signals which do not accomplish the task of recreating the learned trajectory distribution.
We introduce an additional PD controller

$$\mathbf{u}_{PD} = \mathbf{G}\mathbf{e} = \begin{pmatrix} g_{q_1} & g_{\dot{q}_1} & 0 & 0 & \cdots & 0 & 0\\ 0 & 0 & g_{q_2} & g_{\dot{q}_2} & \cdots & 0 & 0\\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & 0 & 0 & \cdots & g_{q_D} & g_{\dot{q}_D}\end{pmatrix}\begin{pmatrix} e_{q_1}\\ e_{\dot{q}_1}\\ \vdots\\ e_{q_D}\\ e_{\dot{q}_D}\end{pmatrix}$$

to compensate for the effects **e** of the prediction errors. The motor signal **u** which is applied to the system is then computed by $\mathbf{u} = \mathbf{u}_{PD} + \mathbf{u}_{MB}$, where $\mathbf{u}_{MB}$ is the motor signal generated by the model-based controller. Besides specifying the proportional and derivative gains $\mathbf{g}_q = [g_{q_1} \ \dots \ g_{q_D}]$ and $\mathbf{g}_{\dot{q}} = [g_{\dot{q}_1} \ \dots \ g_{\dot{q}_D}]$ of the PD controller, it is necessary to infer the effects of the prediction errors. Those effects can be inferred from the conditional distribution over the next system state $\mathbf{y}_{t+dt}$ given the current state $\mathbf{y}_t$. If it holds that the system matrices are predicted correctly, the conditional distribution

$$p^*(\mathbf{y}_{t+dt}|\mathbf{y}_t) = \mathscr{N}\left(\mathbf{y}_{t+dt}|\mathbf{F}\mathbf{y}_t + \mathbf{f}, \mathbf{\Sigma}_s dt\right)$$

should match the distribution obtained by conditioning $p(\mathbf{y}_t, \mathbf{y}_{t+dt}) = \mathscr{N}(\mathbf{y}_t|\boldsymbol{\mu}_t, \mathbf{\Sigma}_t)\mathscr{N}(\mathbf{y}_{t+dt}|\boldsymbol{\mu}_{t+dt}, \mathbf{\Sigma}_{t+dt})$ on $\mathbf{y}_t$

$$p(\mathbf{y}_{t+dt}|\mathbf{y}_t) = \mathscr{N}(\mathbf{y}_{t+dt}|\boldsymbol{\mu}_{t+dt} + \mathbf{C}_t^T\mathbf{\Sigma}_t^{-1}(\mathbf{y}_t - \boldsymbol{\mu}_t), \mathbf{\Sigma}_{t+dt} - \mathbf{C}_t^T\mathbf{\Sigma}_t^{-1}\mathbf{C}_t) = \mathscr{N}(\mathbf{y}_{t+dt}|\boldsymbol{\mu}_{t+dt}^c, \mathbf{\Sigma}_{t+dt}^c)$$

(see appendix A.4 for how to retrieve this distribution). Consequently the system state $\mathbf{y}_{t+dt}$ will have a high probability density $p(\mathbf{y}_{t+dt}|\mathbf{y}_t)$, if the system dynamics are predicted correctly. Considering this, a sensible measure for the effect of prediction errors can be defined as

$$\mathbf{e} = \boldsymbol{\mu}^c_{t+dt} - \mathbf{y}_{t+dt}$$

where $\boldsymbol{\mu}^c_{t+dt}$ is the mean of the likelihood $p(\mathbf{y}_{t+dt}|\boldsymbol{\mu}^c_t)$ conditioned on the old mean $\boldsymbol{\mu}^c_t$ and $\mathbf{y}_{t+dt}$ is the actual state resulting from the application of the motor signals. For $t = 1$, $\boldsymbol{\mu}^c_t$ is initialized with $\mathbf{y}_t$. Note that it is also possible to make use of the covariance matrix of the likelihood, e.g. by multiplying the proportional and derivative gains by some transformation of the probability density $p(\mathbf{y}_{t+dt}|\boldsymbol{\mu}^c_t)$. In the experiments, we will evaluate the effect of this PD controller on the reproduction of the trajectory distribution and its contribution to the motor signal $\mathbf{u}$.

Another problem occurring from wrong predictions of the system matrices are jumps in the motor signals. This can either result from an underestimation of the influence of the motor signals on the system or an overestimation of the forces which act on the system or a combination of both. Those jumps can damage the robot or its environment and hence it is necessary to prevent them. One way to achieve this is to ensure that in every time step t, the velocity of the motor signals generated by the model-based controller

$$\dot{\mathbf{u}}_{MB_t} = \frac{\mathbf{u}_{MB_t} - \mathbf{u}_{MB_{t-dt}}}{dt}$$

does not exceed a certain limit. Naturally if this limit is chosen to low, it will not be possible to recreate the trajectory distribution regardless of the prediction accuracy of the system matrices. In the experiments, the velocity limit was chosen by learning a ProMP on the motor signals of the sample trajectories. The resulting parameter mean $\boldsymbol{\mu}_{\mathbf{w}_u}$ was then multiplied by $\dot{\boldsymbol{\Psi}}$ to retrieve the mean velocity trajectory of the motor signals. The joint limits for the single dimensions of the system were then chosen to be some quantile of its absolute values.

3.1.5 Influence of Prediction Errors

When executing the model-based controller, errors in the predicted system matrices will influence its feedback and feedforward terms $\mathbf{K}$ and $\mathbf{k}$. A more precise knowledge about the relevant terms which influence the feedback and feedforward terms allow for a better evaluation of the learning progress.
Although the calculation of the controller terms depend on all three of the system matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{c}$, only prediction errors of $\mathbf{B}$ and $\mathbf{h} = \mathbf{A}\mathbf{y} + \mathbf{c}$ govern the error in them. This can be seen by rewriting the linear dynamics model as

$$\mathbf{y}_{t+dt} = \mathbf{y}_t + \mathbf{B}dt\mathbf{u} + \mathbf{h}dt$$

and following the derivations presented in [9] to obtain the altered closed-form equations for the controller terms

$$\mathbf{K}^* = \mathbf{B}^\dagger(\dot{\boldsymbol{\Psi}}_t \boldsymbol{\Sigma}_\mathbf{w} \boldsymbol{\Psi}_t^T - \frac{\boldsymbol{\Sigma}_s}{2})\boldsymbol{\Sigma}_t^{-1}$$
$$\mathbf{k}^* = \mathbf{B}^\dagger(\dot{\boldsymbol{\Psi}}_t \boldsymbol{\mu}_\mathbf{w} - \mathbf{B}\mathbf{K}^* \boldsymbol{\Psi}_t \boldsymbol{\mu}_\mathbf{w} - \mathbf{h}).$$

Furthermore the covariance matrix $\boldsymbol{\Sigma}_\mathbf{u}$ of the motor command noise does not change due to the new representation of the linear dynamics model. Now it holds that $\mathbf{K}\mathbf{y} + \mathbf{k} = \mathbf{K}^*\mathbf{y} + \mathbf{k}^*$. The derivation of this result can be reviewed in appendix A.5. Consequently, when evaluating the prediction error of the system matrices with respect to the the terms of the model-based controller, only the errors in $\mathbf{B}$ and $\mathbf{h}$ are of relevance. This also has a graphic interpretation as the controller only needs to know about the forces $\mathbf{h}$ which act on the system in the current state and the influence the motor signals have on those forces, which is expressed by $\mathbf{B}$.

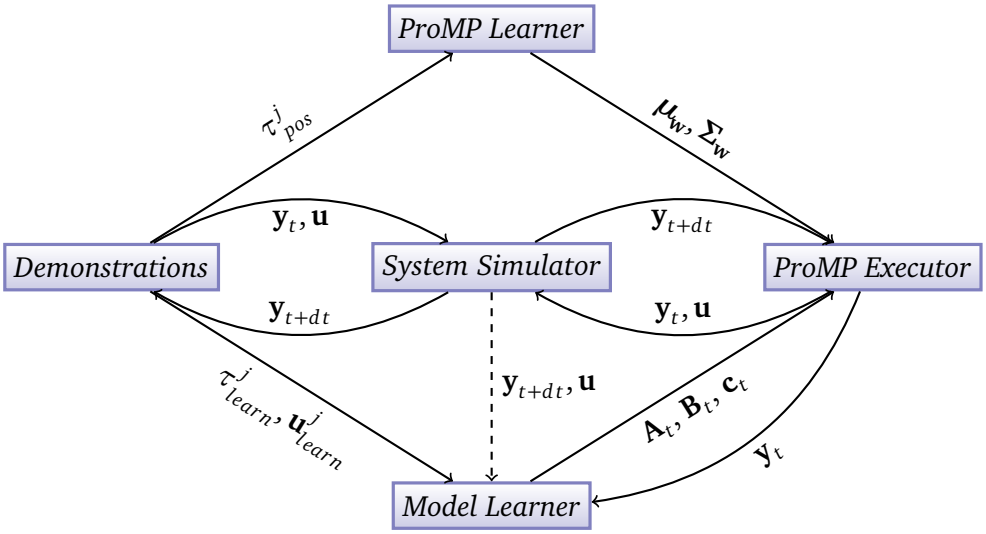

Figure 3.1.: A graphical representation of the interaction between the modules in the software environment which was used to evaluate the learning methods under different scenarios. The solid lines show the interactions which occur during offline learning mode. When online learning is done, the interactions illustrated through the dashed lines occur as well.

3.2 Evaluation Environment

The environment used to evaluate the learning methods was implemented in MATLAB and can be conceptually divided into 5 modules interacting with each other:

- **Demonstrations:** Generates the data needed for learning the trajectory distribution and the linearized dynamics
- **ProMP Learner:** Learns the trajectory distribution as specified in section 2.3
- **Model Learner:** Learns the linearized dynamics by Linear Regression
- **ProMP Executor:** Reproduces the learned trajectory distribution using the model-based controller with the learned system matrices
- **System Simulator:** Simulates the dynamical systems on which the movements should be executed

The learning methods are evaluated by generating sample trajectories and use them to learn the ProMP representation of the movement as well as the linear dynamics model. This representation and the learned model are then used to create trajectories with the model-based controller for ProMPs. The environment allows for two modes of evaluation: an online and an offline setup. In an online setup, the new trajectories which occur during the execution of the ProMP are used to improve the learned model. In an offline setup, this generated data is not taken into account. Figure 3.1 presents, how the single modules interact with each other in both learning setups.
Depending on the dynamical system, learning method and learning setup to evaluate, the single modules are interchanged. In the following paragraphs, implementation details of the single modules are presented as needed.

Demonstrations

For every system under simulation, this module generates different goal trajectories and then uses a PD controller to generate motor signals which lead to a reproduction of those trajectories in the simulator. This results in the sample trajectories τ^j_{learn} and τ^j_{pos} as well as the corresponding motor signals $\mathbf{u}^j_{learn}$.

The goal trajectories are created using cubic splines which interpolate between multiple via-points. In order to generate multiple similar goal trajectories, Gaussian Noise with varying variance is added to the via-points between interpolations.

Model Learner

This part of the test environment evaluates multiple approaches for learning the dynamics of a system. It supports standard Linear Regression as well as Weighted Linear Regression with a distance metric as described in section 3.1.3. During the execution of a ProMP, this module is given the current system state $\mathbf{y}_t$ and estimates the linear dynamics model $\mathbf{A}_t$, $\mathbf{B}_t$ and $\mathbf{c}_t$ by Regression. In online learning setups, this module is also given the new samples $(\dot{\mathbf{y}}_t, (\mathbf{y}_t, \mathbf{u}))$ which are generated during the execution of a ProMP, so that the model can be updated with this new data. Furthermore, learning of a so called inverse dynamics model $\mathbf{u} = f(\mathbf{y}, \ddot{\mathbf{q}})$ using Gaussian Processes is supported as a reference to the presented approach. Details on this reference model are presented in section 4.4.

ProMP Executor

This module executes the learned ProMP by first sampling the starting positions $\mathbf{y}_1$ from the trajectory distribution and then interfering the necessary motor signals as described in section 2.3.2. In an online learning setup, the model-based control law derived from the ProMP is clipped and extended by the PD control law as described in section 3.1.4.

System Simulator

The learning methods were evaluated on different dynamical systems comprising a linear system and non-linear systems such as a pendulum, a double-pendulum and a quad-pendulum. The different pendula systems were described using rigid body dynamics equations, so that non-linearities arise from the gravity and the Coriolis forces. For the single-link pendulum the mass was set to 0.2kg with a length of 0.5m. The links in the double- and quad-link pendula always had a mass of 1kg and a length of 1m. The non-linearities and the increasing dimensionality of the double- and quad-pendulum make the corresponding model learning problem more difficult than for the linear or pendulum system. For the evaluation of the learning progress, the linearized dynamics of the systems need to be calculated. For the non-linear systems, this is done using central differences for a given state $\mathbf{y}$.

4 Experiments

In this chapter, we first present results of offline learning in dynamical systems with increasing complexity and subsequently, we evaluate online learning setups and examine the results of the previously introduced corrective actions. We used the following parameters for training ProMPs using Ridge Regression

- Ridge Regression parameter: $\lambda = 10^{-7}$
- Number of Gaussian Basis Functions: $M = 50$ (spread uniformly over the time interval of the movement)
- Gaussian Basis Function bandwidth: $h = 0.2$

unless explicitly mentioned. For the comparison between learning approaches, the random number generator of MATLAB was set to a fixed value, so that on the same dynamical system, the examined approaches would start execution with the same initial positions $\mathbf{y}_1$. When learning the system matrices, both Linear and Weighted Linear Regression were always used with the prior over the parameter vectors as explained in section 3.1.2. The data from the sample trajectories was transformed as explained in section 3.1.1. For Weighted Linear Regression, the distance metric as shown in section 3.1.3 was used with $k = 100$. Furthermore, for the comparison between the learned trajectory distribution and the one recreated by the model-based controller, always two hundred trajectories were generated.

4.1 ProMP Properties

A relevant implication of ProMPs is that besides the actual prediction accuracy of the linearized dynamics, due to numerical reasons, also the sufficient statistics $\boldsymbol{\mu}_\mathbf{w}$ and $\boldsymbol{\Sigma}_\mathbf{w}$ of the learned trajectory distribution are relevant for the reproduction of this distribution. This can be seen in figure 4.1 which shows executions

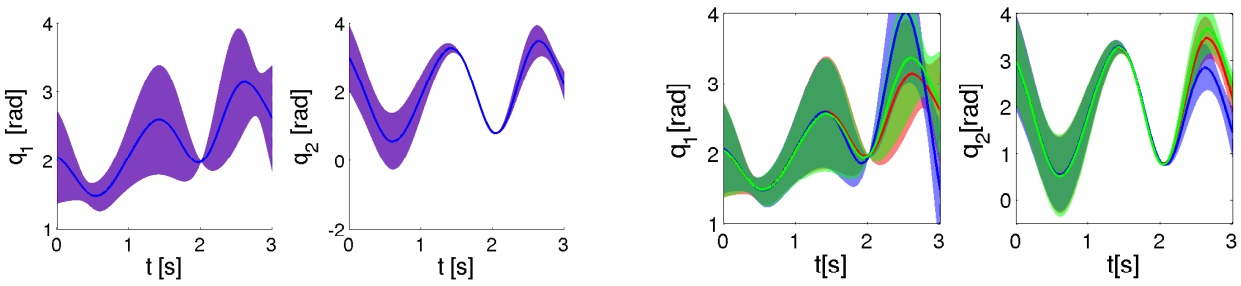

Figure 4.1.: The left plot shows two trajectory distributions over joint positions of a linear system learned from five sample trajectories (red and blue). One of the ProMPs is learned with $M = 50$ Gaussian Basis Functions with bandwidth $h = 0.2$ while the other is learned using $M = 100$ and $h = 0.1$. As can be seen, the learned trajectory distributions match each other. In the right plot, the learned distribution (red) is compared with the distributions created by the model-based controller using the analytical model and the basis function configurations $M = 50, h = 0.2$ (green) and $M = 100, h = 0.1$ (blue). Apparently, although not changing the learned trajectory distribution, the basis configuration can have an influence on the reproduction of this distribution.

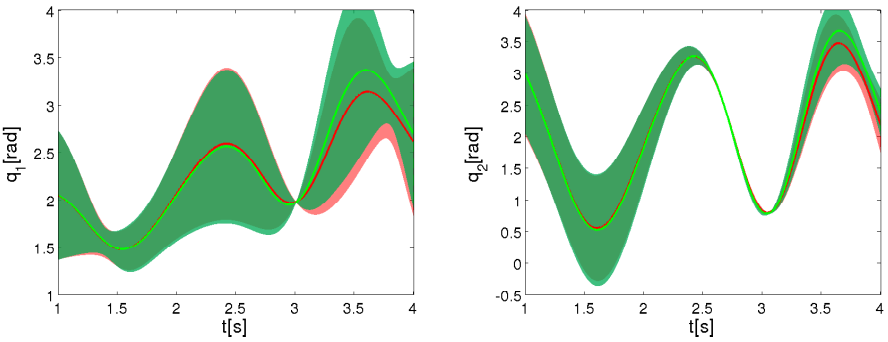

Figure 4.2.: The plot shows the trajectory distribution learned for a linear system from five sample trajectories (red), the distribution created by the model-based controller using the analytical dynamics model (blue) and the distribution created by the model-based controller with learned dynamics using Linear Regression (green).

of two ProMPs trained on the same data for a two-dimensional linear system but with different basis configurations. As a result of this observation, all trajectory distributions generated using the learned system dynamics will also be compared to a baseline, where either the analytical linear dynamics model (for the linear system) or the one obtained by finite differences (for the pendula systems) was used in the model-based controller.

4.2 Offline Learning

The first system on which we evaluated the learning methods was a linear system with two degrees of freedom, where the entries in system matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{c}$ do not depend on the system state $\mathbf{y}$. Hence, the system matrices used by the model-based controller can be learned by standard Linear Regression using all information contained in the sample trajectories. The trajectory distribution resulting from this controller trained with five sample trajectories is presented in figure 4.2.

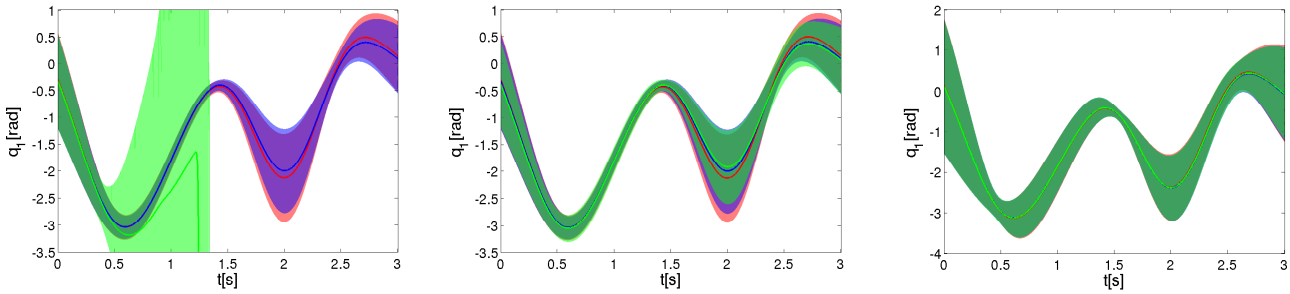

Figure 4.3.: The plots show the trajectory distribution learned for a pendulum system from different amounts of sample trajectories (red), the distribution created by the model-based controller using the analytical dynamics model (blue) and the distributions created by the model-based controller with learned dynamics using Weighted Linear Regression (green). In the left plot, five sample trajectories were given. The plot in the middle also shows results for five sample trajectories, however twelve out of the two hundred executed movements which did not follow the trajectory distribution, were removed. The right plot is the result of fifty sample trajectories.

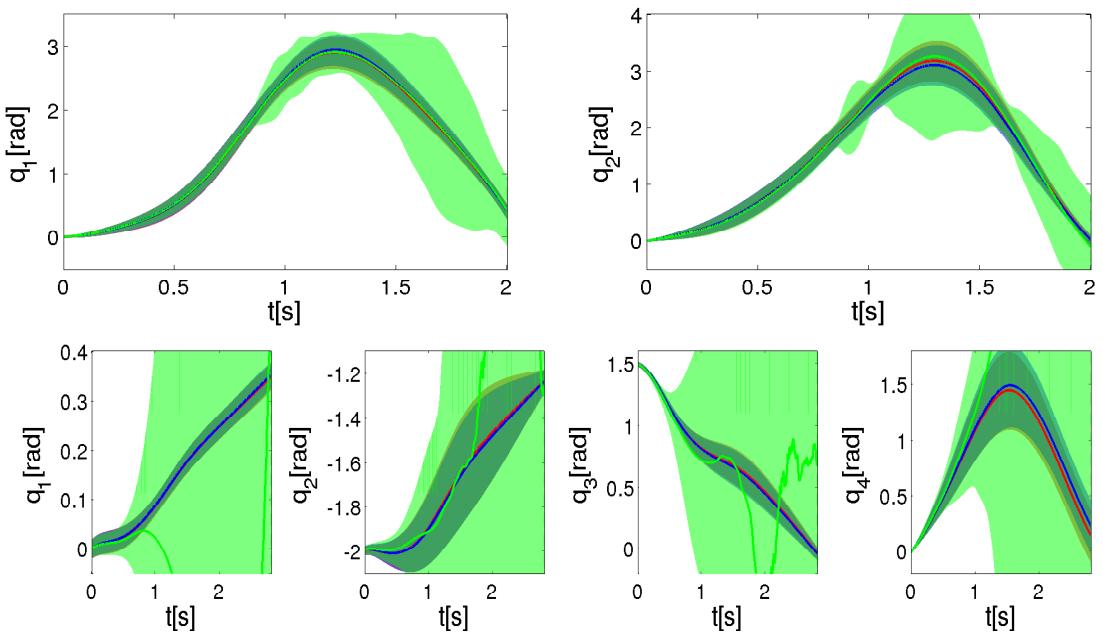

Figure 4.4.: The plots show the trajectory distribution learned for the double- (upper row) and quad-pendulum (lower row) from two hundred sample trajectories (red). The distribution created by the model-based controller using the analytical dynamics model is depicted in blue while the distribution created by the model-based controller with learned dynamics using Weighted Linear Regression is depicted in green.

Naturally, for non-linear systems, such as the pendulum, a global estimate of the system matrices does not suffice to reproduce the learned trajectory distribution, as the system matrices **A**, **B** and **c** are dependent on the system state **y**.

At this point, Weighted Linear Regression is the appropriate tool to learn the state-dependent system matrices. With Weighted Linear Regression it was possible to learn system matrices leading to a reproduction of the trajectory distribution. However, when using five sample trajectories for learning the system matrices, this reproduction was not reliable as some of the executed trajectories were not following the learned distribution. Such problems occur as a result of poor predictions of the system matrices and indicate that not enough sample data similar to the executed trajectories is available. Consequently, increasing the amount of sample trajectories available for learning helps to improve the performance. For the pendulum system, fifty sample trajectories were necessary to achieve a reliable reproduction of the trajectory distribution, as shown in figure 4.3.

For the double- and quad-pendulum, the required number of sample trajectories again increased and even for two-hundred sample trajectories, it was not possible to successfully recreate the learned trajectory distributions. While for the double-pendulum a similar, yet jerky, trajectory distribution could be produced, the results on the quad-pendulum were not even close to the original distribution. Furthermore, the controller generated unstable motor signals for the last steps of the simulation leading to numerical issues with the simulator. The results can be reviewed in figure 4.4, where for the quad-pendulum the last steps of the simulation are left out because of the mentioned problem. However, especially the performance of the learning methods on the double- and quad-pendulum are of interest as for example robotic arms are of similar complexity as those systems. Because of this, the following section presents how the corrective actions presented in chapter 3 facilitate the reproduction of the observed trajectories despite errors in the predictions of the system matrices.

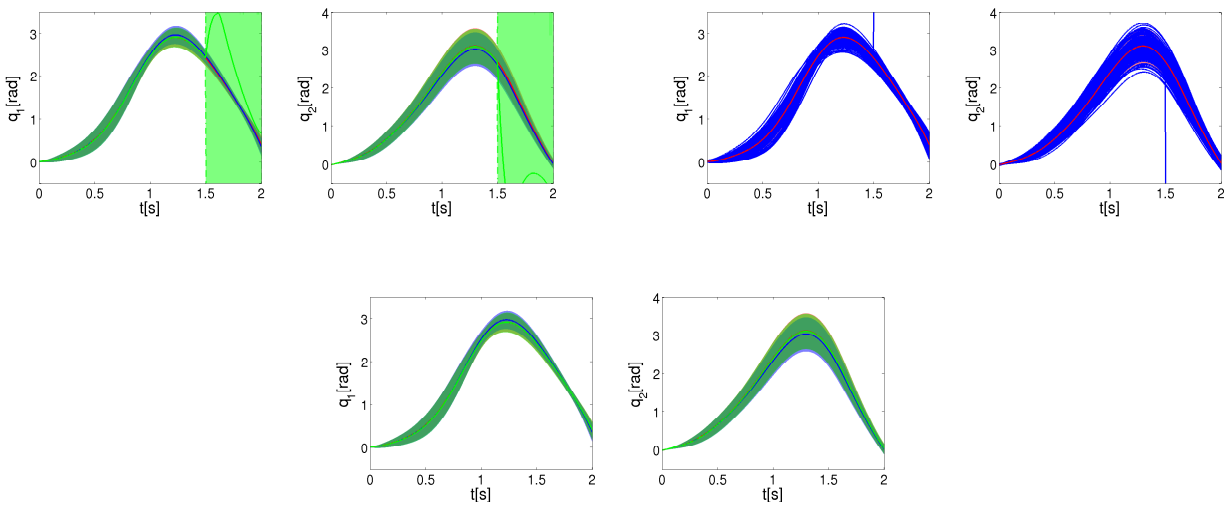

Figure 4.5.: The plots show the trajectory distribution learned for the double-pendulum from five sample trajectories (red), the distribution created by the model-based controller using the analytical dynamics model (blue) and the distribution created by the model-based controller with learned dynamics using Weighted Linear Regression and the PD controller (green). Furthermore single trajectories of the reproduction using learned dynamics are shown by blue lines. In the above plots, no motor signal clipping was used in contrary to the lower plot.

4.3 Online Learning

In online learning setups, we used the PD controller and clipping of the model-based motor signals as presented in section 3.1.4 to compensate for an initially small number of sample trajectories to learn the dynamics from. In the experiments, this approach was able to approximately reproduce the learned trajectory distribution with a much smaller number of sample trajectories than in the offline learning setup. The gains for the PD controller were set to the same or lower values than the ones used to create the sample trajectories and the 0.95-quantile was used to obtain the maximum velocities of the motor signals. The resulting approach was investigated for the double- and quad-pendulum.

1 The relative error between a predicted vector $\mathbf{c}$ and the actual vector $\mathbf{c}^*$ is defined as $\frac{\|\mathbf{c}-\mathbf{c}^*\|_2}{\|\mathbf{c}^*\|_2}$. For matrices $\mathbf{C}$ and $\mathbf{C}^*$ it is defined as the mean of absolute the values in $\mathbf{C}-\mathbf{C}^*$ divided by the mean of the absolute values in $\mathbf{C}^*$

2 The share of $\mathbf{u}_{PD}$ is defined as $\frac{\|\mathbf{u}_{PD}\|_2}{\|\mathbf{u}_{PD}\|_2+\|\mathbf{u}_{MB}\|_2}$

3 This is defined as $\frac{\|\mathbf{u}^c_{MB}-\mathbf{u}_{MB}\|_2}{\|\mathbf{u}_{MB}\|_2}$, where $\mathbf{u}^c_{MB}$ is the possibly clipped motor signal of the model-based controller

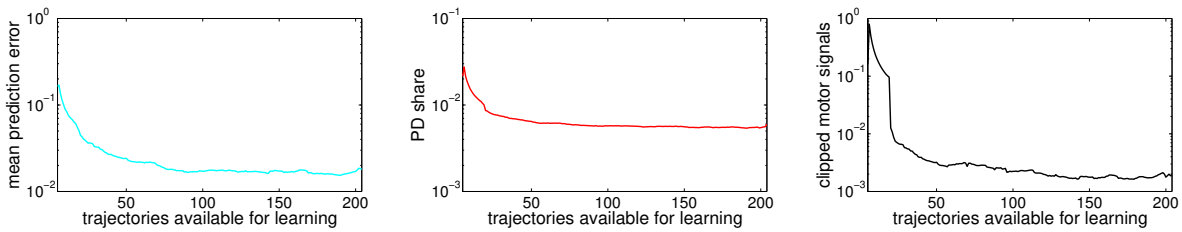

Figure 4.6.: The plots show the mean relative prediction error of $\dot{\mathbf{y}}$[1] (left), the mean share of $\mathbf{u}_{PD}$[2] (middle) and the mean amount of clipped motor signals[3] (right) for the double-pendulum and increasing number of executed trajectories. The plots were created by averaging over ten runs in the online learning setup with different sets of five initial sample trajectories.

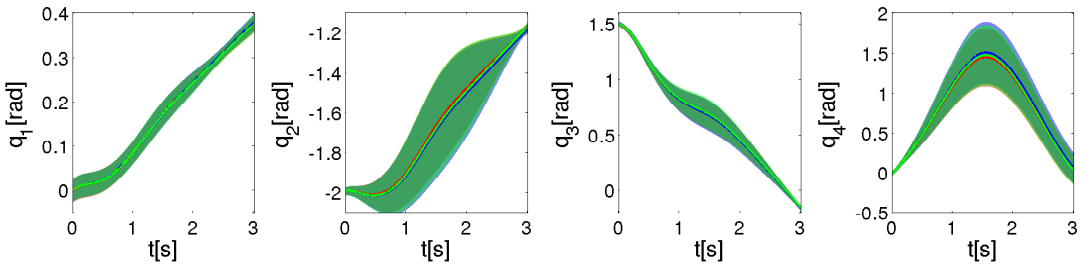

Figure 4.7.: The plots show the trajectory distribution learned for the quad-pendulum from ten sample trajectories (red), the distribution created by the model-based controller using the analytical dynamics model (blue) and the distribution created by the model-based controller with learned dynamics using Weighted Linear Regression with the PD controller and motor signal clipping (green).

When only the PD controller was used, it was possible that at some point during the execution of a trajectory, the influence of the motor signals or the forces acting on the system were that much under- or overestimated, that the controller inferred the need for a motor signal of very large magnitude. As in our setup there existed no limit on the motor signals that could be applied to the dynamical system, this one motor signal was enough to permanently disturb the execution of the movement. This behavior can be prevented through clipping the motor signal to a maximum velocity, as this gives the PD controller enough time to compensate for the incorrect motor signals. The plots in figure 4.5 visualize this problem on the double-pendulum and how clipping the motor signals can prevent it. With both approaches, the reproduction of the learned trajectory distribution on the double-pendulum was possible after only five sample trajectories.
Figure 4.6 shows the learning progress for increasing number of executed trajectories on the double-pendulum. The trend of the mean PD-Share and the amount of clipped motor signals seems to follow the trend of the mean prediction error of $\dot{\mathbf{y}}$.

Finally, figures 4.7 and 4.8 show the result of the controller being applied to the quad-pendulum as well as the corresponding learning progress. Obviously, the same trends in the prediction errors of $\dot{\mathbf{y}}$ as well as the PD-Share and amount of clipped motor signals arise as for the double-pendulum. However, ten sample trajectories were necessary to be able to recreate the learned trajectory distribution for this dynamical system.

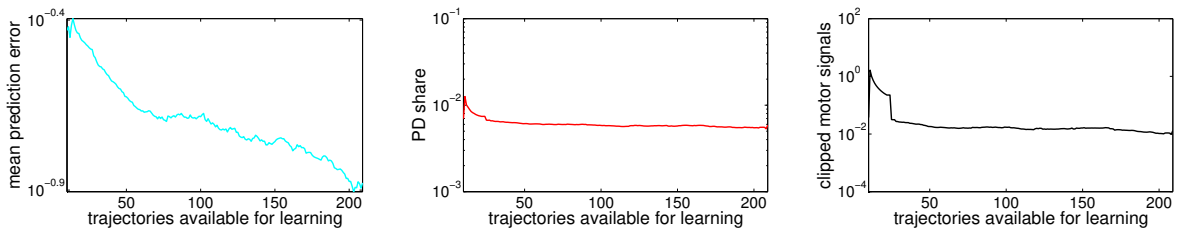

Figure 4.8.: The plots show the mean relative prediction error of $\dot{\mathbf{y}}$ (left), the mean share of $\mathbf{u}_{PD}$ (middle) and the mean amount of clipped motor signals (right) for the quad-pendulum and increasing number of executed trajectories. The plots were created by averaging over ten runs in the online learning setup. For every run, the ten initial sample trajectories were different.

4.4 Comparison to Gaussian Process Regression

To evaluate the performance of the corrective actions presented in chapter 3, a different approach to model-based control with ProMPs was implemented. This approach does not directly compute the motor signals with the model-based controller but uses a learned inverse dynamics model $\mathbf{u} = f(\mathbf{y}, \ddot{\mathbf{q}})$ to compute the motor signals $\mathbf{u}$, which are necessary to reach a desired joint acceleration $\ddot{\mathbf{q}}$ in a given state $\mathbf{y}$. This model is learned with Gaussian Process Regression.
Besides the sample trajectories τ^j_{learn}, also trajectories $\ddot{\mathbf{q}}^j_{learn}$ containing the joint accelerations $\ddot{\mathbf{q}}^j_t$ at the timesteps $t \in [1, \dots, T]$ need to be available for learning. The set of sample data is then defined as

$$\mathscr{D} = \{(\mathbf{u}^j_i, [\mathbf{y}^j_i \ \ddot{\mathbf{q}}^j_{i+1}]), i \in [1, \dots, T-1], j \in [1, \dots, N]\}.$$

In order to employ the learned inverse dynamics model in the model-based controller, alternative system matrices $\mathbf{A}'$, $\mathbf{B}'$ and $\mathbf{c}'$ need to be found so that the resulting controller $\mathbf{K}\mathbf{y}_t + \mathbf{k}$ does not compute the necessary motor signals $\mathbf{u}_{MB}$ to reproduce the trajectory distribution, but the necessary joint accelerations $\ddot{\mathbf{q}}_{des}$ for that. The learned inverse dynamics model is then used to infer the related motor signal. The alternative system matrices can be reviewed in appendix A.6.
In the experiments, a squared exponential kernel was used. The hyperparameters were optimized on a random subset of the data using the Principal Axis Algorithm of the NLopt library [16], which was run until convergence. The size of the subset was set to $250D$, where D is the number of dimensions.
The method based on Gaussian Process Regression was compared to Weighted Linear Regression in combination with the PD controller and motor signal clipping on the double- and quad-pendulum. For the double-pendulum setup, five sample trajectories were given while for the quad-pendulum setup, ten were used. After that, two-hundred trajectories were generated using both methods, without the new trajectories being used for improving the learned dynamics. Figures 4.9 and 4.10 show the results of this comparison. The mean share of $\mathbf{u}_{PD}$ over the executed trajectories was 0.0195 for the double- and 0.0069 for the quad-pendulum. The amount of clipped motor signals was 0.18 and 0.0377 respectively. Interestingly, in the double-pendulum setup there was a higher need for corrective actions than in the quad-pendulum setup, although the double-pendulum is a less complex dynamical system.
Figures 4.9 and 4.10 show that for the same data available for learning, the method based on Gaussian Process Regression was not capable of recreating the learned trajectory distributions, while this was the case for the method using Weighted Linear Regression with the mentioned corrective actions.

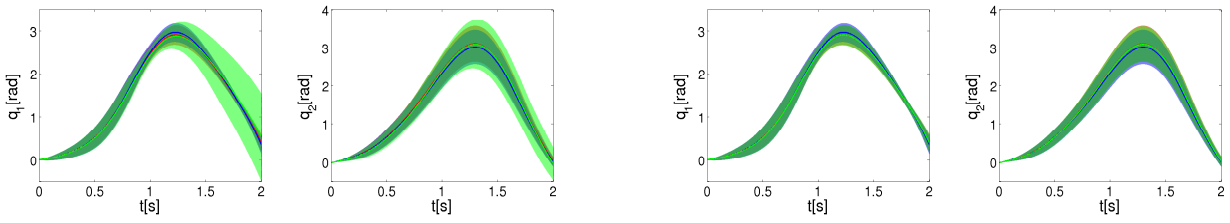

Figure 4.9.: The plots show the recreated trajectory distribution on the double-pendulum when using Weighted Linear Regression with the PD controller and motor signal clipping enabled (right) as well as the presented method using Gaussian Process Regression (left). Five sample trajectories were used for learning and new data was not incorporated. The learned trajectory distribution is shown in red, the one recreated using the model-based controller with the analytical model in blue and the distribution originating from the model-based controller using learned dynamics in green.

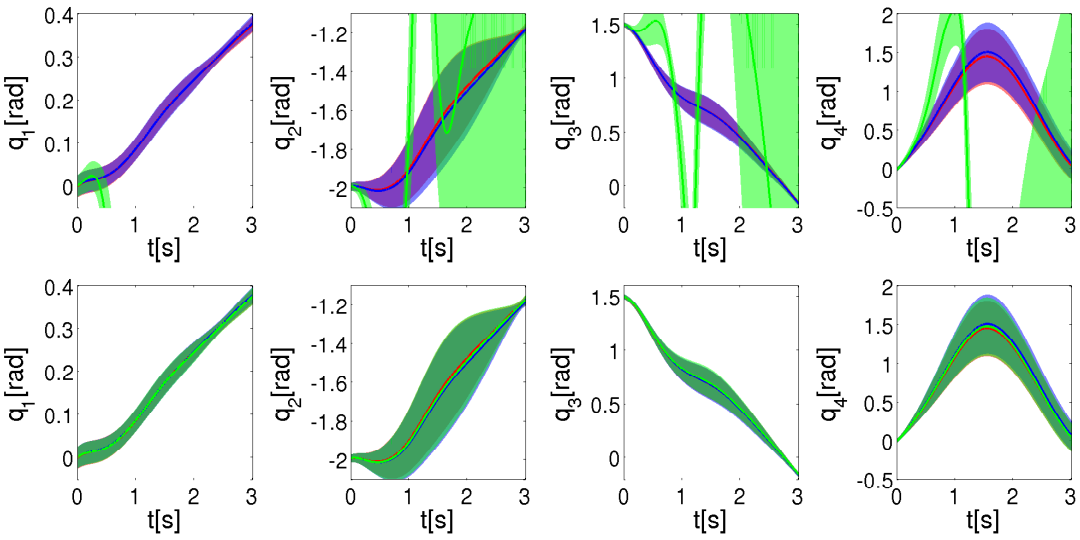

Figure 4.10.: The plots show the recreated trajectory distribution on the quad-pendulum when using Weighted Linear Regression with the PD controller and motor signal clipping enabled (right) as well as the presented method using Gaussian Process Regression (left). Ten sample trajectories were used for learning and new data was not incorporated. The learned trajectory distribution is shown in red, the one recreated using the model-based controller with the analytical model in blue and the distribution originating from the model-based controller using learned dynamics in green.

5 Discussion and Outlook

The experiments showed that with the feature transformation described in 3.1.2, Linear Regression methods can be used to learn linearized system dynamics. However, they also showed that learning linearized system dynamics quickly grows to be a very complex problem for non-linear dynamical systems. Learning those dynamics for ProMPs tend to make this problem even more complex as the resulting model-based controller does not only need to be able to recreate observed sample trajectories but also needs to be able to execute trajectories which were not observed and hence for which only sample data of less similarity is available. Therefore without further aid, using the model-based controller of ProMPs with learned system dynamics is only possible with impractical amounts of sample trajectories for complex systems.

However, the experiments also showed that it is possible to derive a PD controller which compensates for the control error induced by the prediction error of the system matrices. Using this controller in combination with information about the maximally acceptable velocity of the motor signals, it is possible to reduce the necessary amount of sample trajectories to feasible levels. In the experiments, the amount of sample trajectories necessary to recreate the trajectory distribution could not be undercut by a different approach to model-based control using Gaussian Processes.

Interestingly, the changes necessary to the motor signals are on average only very small, as for the online learning results on the double-pendulum the mean PD share over all trajectory executions was only 0.0057 and the mean amount of clipped motor signals only 0.004. For the quad-pendulum, comparable amounts were achieved in the online learning scenario with 0.0056 PD share and 0.022 clipped motor signals. Nevertheless, it is necessary to note that through the introduction of a PD controller, the compliance of the ProMP model-based controller is reduced, as at least in the presented implementation, the PD controller follows a fixed trajectory and hence does not allow for such a big deviation from this trajectory as the completely model-based controller would do.

At this point, using the covariance matrix of the conditioned distribution $p(\mathbf{y}_{t+dt}|\boldsymbol{\mu}_t^c)$ underlying the PD controller may help to increase the compliance. Employing some accuracy measure of the predicted system dynamics in the calculation of the controller gains may increase the compliance furthermore.

Naturally, instead of clipping the velocity of the motor signals it may be possible to ensure a smooth change of the entries in the linearized system dynamics and with that a smooth change in the motor signals through additional constraints on the optimization problem underlying the employed learning

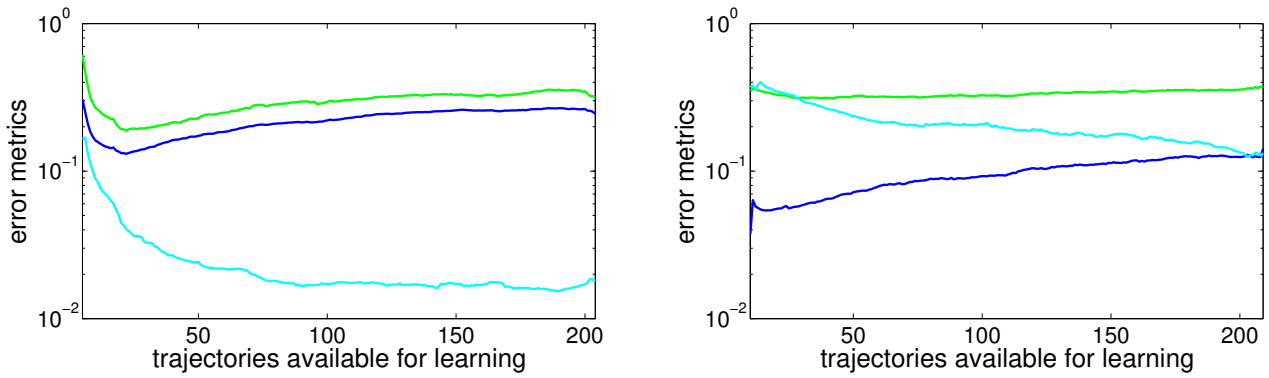

Figure 5.1.: The plots show the mean relative prediction error of $\dot{\mathbf{y}}$ (cyan), $\mathbf{B}$ (blue) and $\mathbf{h}$ (green) for increasing number of executed trajectories on the double-pendulum (left) and quad-pendulum (right).

methods. The predictions of the system matrices at timestep $t + dt$ in the experiments were independent of the ones made in t, although in practice the predicted values are connected in some way.
An interesting observation of the experiments for the online learning setup is that while the trend of the mean prediction error of $\mathbf{B}$ and $\mathbf{h}$ is even slightly increasing with increasing number of executed trajectories, the mean prediction error of $\dot{\mathbf{y}}$ decreases. This effect probably arises due to a combination of two actualities. For one, the cost function which is optimized by Weighted Linear Regression is defined as the maximization of the likelihood of $\dot{\mathbf{y}}$ and not of $\mathbf{B}$ or $\mathbf{h}$, as the latter quantities cannot be observed. On the other hand, except for the linear system, the simulators did not compute the next state $\mathbf{y}_{t+dt}$ by applying the linearized system dynamics to the current state $\mathbf{y}_t$ for the time interval dt but by doing multiple intermediate steps in between. Because of this, the linearized system dynamics may not be the optimal parameters with respect to the cost function of the Regression method. Figure 5.1 visualizes the mentioned observations for the double- and quad-pendulum. Keeping the discrepancy between the prediction errors in mind, a topic which could be of interest to further research is whether and if so why the clipping amount and the PD share are correlated with the prediction error of $\dot{\mathbf{y}}$, although the motor signals are calculated using the predictions of the system matrices. One possible explanation for that may be that if underestimating the influence of the motor signals on the system goes along with the underestimation of forces applying to the system, the individual errors are canceled out. It would be interesting to see, whether those observations can be backed by some bound on the motor signal error induced by errors in the prediction of $\mathbf{B}$ and $\mathbf{h}$.
Finally, evaluating the presented online learning approach on a real robot could show whether it is useful in real-world scenarios.

Bibliography

[1] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2012.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., second ed., 2006.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016.

[4] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, p. 0278364913495721, 2013.

[5] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive processing*, vol. 12, no. 4, pp. 319–340, 2011.

[6] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *Experimental Robotics IX*, pp. 363–372, Springer, 2006.

[7] J. Peters, K. Mülling, and Y. Altun, "Relative entropy policy search.," in *AAAI*, Atlanta, 2010.

[8] S. Schaal, "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics," in *Adaptive Motion of Animals and Machines*, pp. 261–280, Springer, 2006.

[9] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in neural information processing systems*, pp. 2616–2624, 2013.

[10] C. Bracegirdle, "Bayes' theorem for gaussians," 2010.

[11] C. E. Rasmussen, "Gaussian processes for machine learning," 2006.

[12] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning,"

[13] D. Ormoneit and T. Hastie, "Optimal kernel shapes for local linear regression," in *NIPS*, pp. 540–546, 1999.

[14] J.-A. Ting, M. Kalakrishnan, S. Vijayakumar, and S. Schaal, "Bayesian kernel shaping for learning control," in *Advances in neural information processing systems*, pp. 1673–1680, 2009.

[15] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *Journal of the ACM (JACM)*, vol. 45, no. 6, pp. 891–923, 1998.

[16] "NLopt homepage." `http://ab-initio.mit.edu/wiki/index.php/NLopt`. Accessed: 2016-10-10.

[17] K. B. Petersen, M. S. Pedersen, *et al.*, "The matrix cookbook," *Technical University of Denmark*, vol. 7, p. 15, 2008.

A Derivations and Definitions

A.1 Maximum Likelihood Estimation for Gaussian Distributions

Using results from matrix calculus presented in [17], the derivative of $ln(L(\mathscr{D}, \boldsymbol{\theta}))$ with respect to $\boldsymbol{\Sigma}$ can be expressed through

$$\begin{aligned}
\frac{\partial ln(L(\mathscr{D}, \boldsymbol{\theta}))}{\partial \boldsymbol{\Sigma}} &= -\frac{ND}{2}\frac{\partial ln(2\pi)}{\partial \boldsymbol{\Sigma}} - \frac{N}{2}\frac{\partial ln(|\boldsymbol{\Sigma}|)}{\partial \boldsymbol{\Sigma}} - \frac{1}{2}\sum_{i=1}^{N}\frac{\partial (\mathbf{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu})}{\partial \boldsymbol{\Sigma}} \\
&= -\frac{N}{2}\frac{\partial ln(|\boldsymbol{\Sigma}|)}{\partial \boldsymbol{\Sigma}} - \frac{1}{2}\sum_{i=1}^{N}\frac{\partial (\mathbf{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu})}{\partial \boldsymbol{\Sigma}} \\
&= -\frac{N}{2}\frac{1}{|\boldsymbol{\Sigma}|}|\boldsymbol{\Sigma}|(\boldsymbol{\Sigma}^{-1})^T + \frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{\Sigma}^{-1})^T(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T(\boldsymbol{\Sigma}^{-1})^T \\
&= -\frac{N}{2}\boldsymbol{\Sigma}^{-1} + \frac{1}{2}\boldsymbol{\Sigma}^{-1}\left(\sum_{i=1}^{N}(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T\right)\boldsymbol{\Sigma}^{-1}.
\end{aligned}$$

This derivative is set to zero and the resulting equation solved for $\boldsymbol{\Sigma}$

$$\begin{aligned}
0 &= -\frac{N}{2}\boldsymbol{\Sigma}^{-1} + \frac{1}{2}\boldsymbol{\Sigma}^{-1}\left(\sum_{i=1}^{N}(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T\right)\boldsymbol{\Sigma}^{-1} \\
\Leftrightarrow N\boldsymbol{\Sigma}^{-1} &= \boldsymbol{\Sigma}^{-1}\left(\sum_{i=1}^{N}(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T\right)\boldsymbol{\Sigma}^{-1} \\
\Leftrightarrow \boldsymbol{\Sigma}N &= \sum_{i=1}^{N}(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T \\
\Leftrightarrow \boldsymbol{\Sigma} &= \frac{\sum_{i=1}^{N}(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T}{N}
\end{aligned}$$

For $\boldsymbol{\mu}$, the derivative of the cost function is expressed by

$$\begin{aligned}
\frac{\partial ln(L(\mathscr{D}, \boldsymbol{\theta}))}{\partial \boldsymbol{\mu}} &= -\frac{1}{2}\sum_{i=1}^{N}\frac{\partial (\mathbf{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu})}{\partial \boldsymbol{\mu}} \\
&= \sum_{i=1}^{N}\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}).
\end{aligned}$$

and hence the optimal parameters with regards to the optimization problem are given by

$$\begin{aligned} 0 &= \sum_{i=1}^{N} \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}) \\ \Leftrightarrow \sum_{i=1}^{N} \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} &= \sum_{i=1}^{N} \boldsymbol{\Sigma}^{-1}\mathbf{x}_i \\ \Leftrightarrow \sum_{i=1}^{N} \boldsymbol{\mu} &= \sum_{i=1}^{N} \mathbf{x}_i \\ \Leftrightarrow \boldsymbol{\mu} &= \frac{\sum_{i=1}^{N} \mathbf{x}_i}{N}. \end{aligned}$$

A.2 Least Squares / Maximum Likelihood Regression

A.2.1 Optimal Parameters

To ease comparisons to other Regression methods, it is preferable to rewrite $J_{LS}(\mathscr{D}, \boldsymbol{\theta})$ as a vector product

$$J_{LS}(\mathscr{D}, \boldsymbol{\theta}) = \frac{1}{2}(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta})^T(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta}), \quad \mathbf{y} = [y_1 \dots y_N]^T, \quad \boldsymbol{\Phi} = [\phi(\mathbf{x}_1) \dots \phi(\mathbf{x}_N)]^T$$

whose derivative with respect to $\boldsymbol{\theta}$ can now again be calculated using the results presented in [17]

$$\frac{\partial J(\mathscr{D}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{1}{2} 2(-\boldsymbol{\Phi})^T(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta}) = -\boldsymbol{\Phi}^T(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta}).$$

The closed form expression of $\boldsymbol{\theta}$ which minimizes $J_{LS}(\mathscr{D}, \boldsymbol{\theta})$ can be retrieved by setting the above derivative to zero and solving for $\boldsymbol{\theta}$

$$\begin{aligned} \mathbf{0} &= -\boldsymbol{\Phi}^T(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta}) \\ \Leftrightarrow \mathbf{0} &= -\boldsymbol{\Phi}^T\mathbf{y} + \boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{\theta} \\ \Leftrightarrow \boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{\theta} &= \boldsymbol{\Phi}^T\mathbf{y} \\ \Leftrightarrow \boldsymbol{\theta} &= (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathbf{y} \end{aligned}$$

A.2.2 Maximum Likelihood Regression

Just as in section 2.1.3, it is easier to calculate $\underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ ln(J_{ML}(\mathscr{D}, \boldsymbol{\theta})) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ ln(\mathscr{N}(\mathbf{y}|\boldsymbol{\Phi}\boldsymbol{\theta}, \sigma^2\mathbf{I}))$. Reformulating this objective

$$\begin{aligned} \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ ln(J_{ML}(\mathscr{D}, \boldsymbol{\theta})) &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ ln(\mathscr{N}(\mathbf{y}|\boldsymbol{\Phi}\boldsymbol{\theta}, \sigma^2\mathbf{I})) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ ln\left(\frac{1}{(2\pi)^{\frac{K}{2}}}\frac{1}{|\sigma^2\mathbf{I}|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta})^T\frac{1}{\sigma^2}\mathbf{I}(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta})\right)\right) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\left(-\frac{K}{2}ln(2\pi) - \frac{1}{2}ln(|\sigma^2\mathbf{I}|) - \frac{1}{2}(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta})^T\frac{1}{\sigma^2}\mathbf{I}(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta})\right) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\left(-\frac{1}{2\sigma^2}(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta})^T(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta})\right) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left(\frac{1}{2}(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta})^T(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta})\right) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\ J_{LS}(\mathscr{D}, \boldsymbol{\theta}) \end{aligned}$$

shows that the optimization of both objective functions lead to the same result.

A.3 Weighted Least Squares / Weighted Maximum Likelihood Regression

A.3.1 Optimal Parameters

As for the basic Least Squares Regression, it is again useful to rewrite the cost function to consist of vector and matrix multiplications

$$J_{WLS}(\mathscr{D},\boldsymbol{\theta},\mathbf{x}_q)=\frac{1}{2}(\mathbf{y}-\boldsymbol{\Phi}\boldsymbol{\theta})^T\mathbf{W}(\mathbf{y}-\boldsymbol{\Phi}\boldsymbol{\theta}),\quad \mathbf{W}=diag(w_1,\ldots,w_N)=\begin{pmatrix} w_1 & 0 & \ldots & 0\\ 0 & w_2 & \ldots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \ldots & w_N\end{pmatrix}.$$

Again the optimal parameters are retrieved by computing $\underset{\theta}{\text{argmin}}\, J_{WLS}(\mathscr{D},\boldsymbol{\theta},\mathbf{x}_q)$. The needed derivative for this is given by

$$\frac{\partial J_{WLS}(\mathscr{D},\boldsymbol{\theta},\mathbf{x}_q)}{\partial\boldsymbol{\theta}}=-\boldsymbol{\Phi}^T\mathbf{W}(\mathbf{y}-\boldsymbol{\Phi}\boldsymbol{\theta})$$

which is again set to zero and solved for $\boldsymbol{\theta}$ to derive the parameters which minimize the cost function

$$\begin{aligned}\mathbf{0}&=-\boldsymbol{\Phi}^T\mathbf{W}(\mathbf{y}-\boldsymbol{\Phi}\boldsymbol{\theta})\\ \Leftrightarrow\ \mathbf{0}&=-\boldsymbol{\Phi}^T\mathbf{W}\mathbf{y}+\boldsymbol{\Phi}^T\mathbf{W}\boldsymbol{\Phi}\boldsymbol{\theta}\\ \Leftrightarrow\ \boldsymbol{\theta}&=(\boldsymbol{\Phi}^T\mathbf{W}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathbf{W}\mathbf{y}.\end{aligned}$$

A.3.2 Weighted Maximum Likelihood Regression

The objective $\underset{\boldsymbol{\theta}}{\text{argmax}}\, ln(J_{WML}(\mathscr{D},\boldsymbol{\theta},\mathbf{x}_q))$ can be reformulated as

$$\begin{aligned}\underset{\boldsymbol{\theta}}{\text{argmax}}\, ln(J_{WML}(\mathscr{D},\boldsymbol{\theta},\mathbf{x}_q))&=\underset{\boldsymbol{\theta}}{\text{argmax}}\, ln(\mathscr{N}(\mathbf{y}|\boldsymbol{\Phi}\boldsymbol{\theta},\mathbf{W}^{-1}))\\ &=\underset{\boldsymbol{\theta}}{\text{argmax}}\, ln\left(\frac{1}{(2\pi)^{\frac{K}{2}}}\frac{1}{|\mathbf{W}^{-1}|^{\frac{1}{2}}}exp\left(-\frac{1}{2}(\mathbf{y}-\boldsymbol{\Phi}\boldsymbol{\theta})^T\mathbf{W}(\mathbf{y}-\boldsymbol{\Phi}\boldsymbol{\theta})\right)\right)\\ &=\underset{\boldsymbol{\theta}}{\text{argmin}}\left(\frac{1}{2}(\mathbf{y}-\boldsymbol{\Phi}\boldsymbol{\theta})^T\mathbf{W}(\mathbf{y}-\boldsymbol{\Phi}\boldsymbol{\theta})\right)=\underset{\boldsymbol{\theta}}{\text{argmin}}\, J_{WLS}(\mathscr{D},\boldsymbol{\theta},\mathbf{x}_q).\end{aligned}$$

Consequently, both maximizing J_{WML} and minimizing J_{WLS} leads to the same result.

A.4 Conditioned State Distribution

According to [9], the joint distribution $p(\mathbf{y}_t,\mathbf{y}_{t+dt})$ takes the form

$$\mathscr{N}\left(\begin{pmatrix}\mathbf{y}_t\\ \mathbf{y}_{t+dt}\end{pmatrix}\middle|\begin{pmatrix}\boldsymbol{\mu}_t\\ \boldsymbol{\mu}_{t+dt}\end{pmatrix},\begin{pmatrix}\boldsymbol{\Sigma}_t & \mathbf{C}_t\\ \mathbf{C}_t^T & \boldsymbol{\Sigma}_{t+dt}\end{pmatrix}\right)=\mathscr{N}\left(\begin{pmatrix}\mathbf{y}_{t+dt}\\ \mathbf{y}_t\end{pmatrix}\middle|\begin{pmatrix}\boldsymbol{\mu}_{t+dt}\\ \boldsymbol{\mu}_t\end{pmatrix},\begin{pmatrix}\boldsymbol{\Sigma}_{t+dt} & \mathbf{C}_t^T\\ \mathbf{C}_t & \boldsymbol{\Sigma}_t\end{pmatrix}\right)$$

where $\mathbf{C}_t=\boldsymbol{\Psi}_t\boldsymbol{\Sigma}_{\mathbf{w}\boldsymbol{\Psi}^T_{t+dt}}$. Using a result from [10] it is possible to express the conditioned distribution $p(\mathbf{y}_{t+dt},\mathbf{y}_t)$ through

$$p(\mathbf{y}_{t+dt}|\mathbf{y}_t)=\mathscr{N}(\mathbf{y}_{t+dt}|\boldsymbol{\mu}_{t+dt}+\mathbf{C}_t^T\boldsymbol{\Sigma}_t^{-1}(\mathbf{y}_t-\boldsymbol{\mu}_t),\boldsymbol{\Sigma}_{t+dt}-\mathbf{C}_t^T\boldsymbol{\Sigma}_t^{-1}\mathbf{C}_t).$$

A.5 Altered Model-Based Controller

For $\mathbf{K}$ and $\mathbf{k}$ as defined in equations 2.2 and 2.3 and

$$\mathbf{K}^* = \mathbf{B}^\dagger(\dot{\mathbf{\Psi}}_t \mathbf{\Sigma}_\mathbf{w} \mathbf{\Psi}_t^T - \frac{\mathbf{\Sigma}_s}{2})\mathbf{\Sigma}_t^{-1}$$
$$\mathbf{k}^* = \mathbf{B}^\dagger(\dot{\mathbf{\Psi}}_t \boldsymbol{\mu}_\mathbf{w} - \mathbf{B}\mathbf{K}^* \mathbf{\Psi}_t \boldsymbol{\mu}_\mathbf{w} - \mathbf{h}).$$

it holds that

$$\begin{aligned}
\mathbf{K}\mathbf{y} + \mathbf{k} &= (\mathbf{K}^* - \mathbf{B}^\dagger\mathbf{A})\mathbf{y} + \mathbf{k} \\
&= (\mathbf{K}^* - \mathbf{B}^\dagger\mathbf{A})\mathbf{y} + (\mathbf{B}^\dagger(\dot{\mathbf{\Psi}}_t \boldsymbol{\mu}_\mathbf{w} - \mathbf{B}\mathbf{K}\mathbf{\Psi}_t \boldsymbol{\mu}_\mathbf{w} - \mathbf{h}) - \mathbf{B}^\dagger\mathbf{A}\mathbf{\Psi}_t \boldsymbol{\mu}_\mathbf{w} + \mathbf{B}^\dagger\mathbf{A}\mathbf{y}) \\
&= (\mathbf{K}^* - \mathbf{B}^\dagger\mathbf{A})\mathbf{y} + (\mathbf{B}^\dagger(\dot{\mathbf{\Psi}}_t \boldsymbol{\mu}_\mathbf{w} - \mathbf{B}(\mathbf{K}^* - \mathbf{B}^\dagger\mathbf{A})\mathbf{\Psi}_t \boldsymbol{\mu}_\mathbf{w} - \mathbf{h}) - \mathbf{B}^\dagger\mathbf{A}\mathbf{\Psi}_t \boldsymbol{\mu}_\mathbf{w} + \mathbf{B}^\dagger\mathbf{A}\mathbf{y}) \\
&= \mathbf{K}^*\mathbf{y} + \mathbf{B}^\dagger(\dot{\mathbf{\Psi}}_t \boldsymbol{\mu}_\mathbf{w} - \mathbf{B}\mathbf{K}^*\mathbf{\Psi}_t \boldsymbol{\mu}_\mathbf{w} - \mathbf{h}) + \mathbf{B}^\dagger\mathbf{A}\mathbf{\Psi}_t \boldsymbol{\mu}_\mathbf{w} - \mathbf{B}^\dagger\mathbf{A}\mathbf{\Psi}_t \boldsymbol{\mu}_\mathbf{w} + \mathbf{B}^\dagger\mathbf{A}\mathbf{y} - \mathbf{B}^\dagger\mathbf{A}\mathbf{y} \\
&= \mathbf{K}^*\mathbf{y} + \mathbf{k}^*.
\end{aligned}$$

A.6 Alternative System Matrices

To make the model-based controller compute $\ddot{\mathbf{q}}$ instead of $\mathbf{u}$, the system matrices $\mathbf{A}'$, $\mathbf{B}'$ and $\mathbf{c}'$ need to be defined as

$$\mathbf{A}' = \begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & 0 & \cdots & 0 \end{pmatrix}, \quad \mathbf{B}' = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 1 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}.$$