



# **Generalization of optimal motion trajectories for a biped walking machine based on machine learning**

Dietrich Trautmann



**MASTER'S THESIS**

**GENERALIZATION OF OPTIMAL  
MOTION TRAJECTORIES FOR A BIPED  
WALKING MACHINE BASED ON  
MACHINE LEARNING**

Freigabe:

Der Bearbeiter:

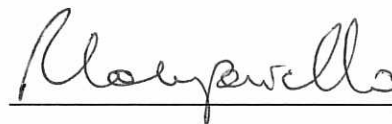
Dietrich Trautmann

Unterschriften



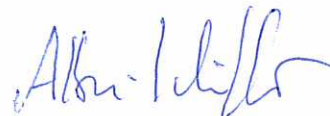
Betreuer:

Roberto Lampariello

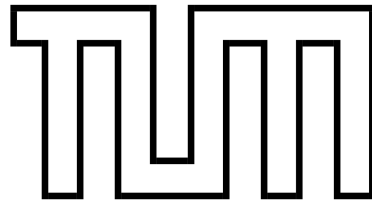


Der Institutsdirektor

Dr. Alin Albu-Schäffer



Dieser Bericht enthält 66 Seiten, 34 Abbildungen und 4 Tabellen



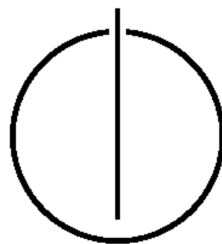
FAKULTÄT FÜR INFORMATIK  
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, Intelligence

**Generalization of optimal motion trajectories  
for a biped walking machine  
based on machine learning**

**Verallgemeinerung optimierter Trajektorien  
für den TORO Laufroboter  
durch maschinelles Lernen**

Author: Dietrich Trautmann  
Supervisor: Prof. Dongheui Lee, Ph.D.  
Advisor: Roberto Lampariello  
Date: 15.04.2015





Ich versichere, dass ich diese Master's Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I confirm that this master's thesis is my own work and I have documented all sources and material used.

München, 15.04.2015

Dietrich Trautmann



## **Abstract**

Optimal walking trajectories are essential for bipedal walking. Recent developments enabled the use of motion planner based on nonlinear optimization. They are however, not directly applicable in real-time tasks due to a high computation time. Therefore, a task space consisting of the stride-length and a step time is used to precompute corresponding trajectory parameters in a certain range regarding a cost function. The resulting trajectories define an optimal joint space motion. The mapping from the task space to the joint space is generalized with multiple machine learning methods. A parametrization of every method was determined to represent the underlying model of the data as good as possible, without overfitting. Finally, the performance regarding the accuracy and runtime and the evaluation of the cost value and constraint violation in the walking tasks is discussed.





## **Danksagung**

Ich bedanke mich sehr herzlich bei Prof. Dongheui Lee von der Technischen Universität München (TUM), sowie Roberto Lampariello und Alexander Werner vom Deutschen Zentrum für Luft- und Raumfahrt (DLR) für das ermöglichen der Master's Thesis als auch für die vielen Kommentare und Diskussionen in Besprechungen.

Ein besonderer Dank gilt meinen Eltern Svetlana und Eduard Trautmann, meinen Bruder Christian Trautmann, meiner Oma Gertrud Trautmann sowie meiner Freundin Alena Moiseeva für die moralische Unterstützung während der Arbeit.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Humanoid Robot Locomotion . . . . .	2
1.2	Related Work . . . . .	3
1.3	Motivation . . . . .	4
1.4	Contribution . . . . .	4
1.5	Outline . . . . .	4
<b>2</b>	<b>Problem Statement and Optimization Problem</b>	<b>7</b>
2.1	Problem Statement . . . . .	7
2.2	Optimization Problem . . . . .	8
2.3	Data Set Generation . . . . .	9
<b>3</b>	<b>Applied Machine Learning Methods</b>	<b>11</b>
3.1	Machine Learning Introduction . . . . .	11
3.2	Machine Learning Methods . . . . .	15
3.2.1	N-Nearest Neighbors . . . . .	15
3.2.2	Regression Tree . . . . .	17
3.2.3	Linear and Polynomial Regression . . . . .	19
3.2.4	Support Vector Regression . . . . .	21
3.2.5	Gaussian Process Regression . . . . .	25
3.3	Clustering with Gaussian Mixture Models . . . . .	28
<b>4</b>	<b>Application of Methods on Data</b>	<b>31</b>
4.1	Methods Parameter Determination . . . . .	31
4.2	Full Task Space Prediction . . . . .	37
4.3	Data Clustering . . . . .	38
<b>5</b>	<b>Results</b>	<b>41</b>
5.1	Best Parametrization . . . . .	41
5.2	Clustering of the Torque Cost Data Set . . . . .	43
5.3	Machine Learning Methods Performance . . . . .	45
5.3.1	Velocity Cost Data Set . . . . .	45
5.3.2	Torque Cost Data Set . . . . .	45
5.4	Evaluation of the Predicted Trajectory Parameters . . . . .	47

5.4.1	Velocity Cost Data Set . . . . .	47
5.4.2	Torque Cost Data Set . . . . .	47
5.5	Simulation with Predicted Parameter . . . . .	49
<b>6</b>	<b>Conclusion &amp; Outlook</b>	<b>51</b>
	<b>List of Figures</b>	<b>53</b>
	<b>Bibliography</b>	<b>55</b>

# Chapter 1

## Introduction

Humanoid robots are getting more attention than ever. The technological development allows already not only the creation of robots with humanlike appearance, but also with humanlike capabilities. This helps researcher to gain more understanding about us humans, but offers also the possibility to use humanoid robots to serve for human needs where it is either dangerous or impossible to work for humans or to increase the productivity in the form of service-robots in households.

Most of the current service-robots investigated in research projects have an upper humanlike body, a head with cameras and two arms with gripper or multifingered hands, but often only a mobile platform with wheels. Examples are the Rollin' Justin [BWS<sup>+</sup>09] at the DLR, the PR2<sup>1</sup> from Willow Garage, Baxter<sup>2</sup> from Rethink Robotics or Pepper<sup>3</sup> from Aldebaran. These systems are limited by their stationary or wheeled platform in environments built for humans and hence aren't capable of achieving a task where they have to overcome a step or a gap. Using biped walker, general purpose service-robots are possible. The development started with the first actuated biped walker WABIAN and WABIAN-II [OAS<sup>+</sup>06] at the University of Waseda and since then a lot of humanoid walking robots were created. Examples of humanoid walking machines are the HRP-2 [HKK<sup>+</sup>04] from AIST, LOLA [LBU09] from the Technische Universität München (TUM), ASIMO<sup>4</sup> from Honda, Petman and Atlas from Boston Dynamics<sup>5</sup> and TORO [EWO<sup>+</sup>], the torque-controlled humanoid robot from the German Aerospace Center (DLR). TORO was also the target platform for the task in this thesis, see Fig. 1.1. The advancement in the field of humanoid robots in recent years triggered the attention of several big companies, with resulting multiple acquisitions. Softbank acquired Aldebaran and Google bought Boston Dynamics and several other smaller robotic related companies. These investments promise an impact on the whole field of humanoid robotics.

---

<sup>1</sup><https://www.willowgarage.com/pages/pr2/overview>

<sup>2</sup><http://www.rethinkrobotics.com/baxter/>

<sup>3</sup><http://www.aldebaran.com/en/a-robots/who-is-pepper>

<sup>4</sup><http://asimo.honda.com>

<sup>5</sup>[http://www.bostondynamics.com/robot\\_Atlas.html](http://www.bostondynamics.com/robot_Atlas.html)

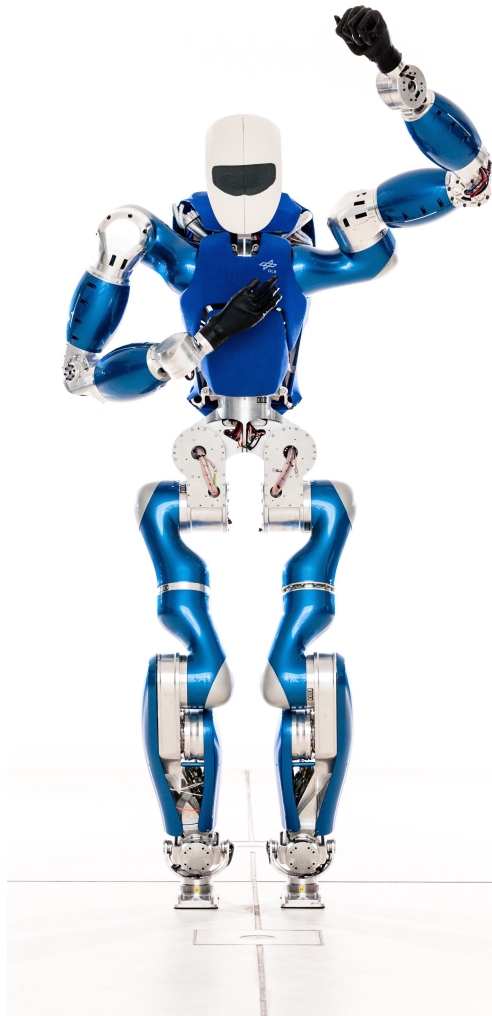


Figure 1.1: TORO, the torque-controlled humanoid robot (DLR)

## 1.1 Humanoid Robot Locomotion

Walking is for humans an every day task and is accomplished without any hassle. The mentioned importance of bipedal walking for humanoid robots in environments created for humans is a popular research topic, since the robustness of the human locomotion is not completely understood to this day. Therefore, this section will present the theory and vocabulary used in this field since it is important for the task targeted in this thesis.

A regular walking step for humans and humanoids (short for humanoid robots) is divided into two phases, the single-support phase with only one foot contact at a time to the ground and the double-support phase with both feet on the ground. The

feet of a humanoid have therefore several contact points with the ground. Taking a convex hull of all contact points in the single- and double-support phase results in a supportpolygon for the current state of the robot.

The supportpolygon is important for statical walking. Projecting the center of mass (CoM) as one point on the ground, the condition for statical walking implies that as long as the projected CoM point stays within the supportpolygon, a statical stable gait is possible. To allow humanlike gaits, the CoM must be able to leave the supportpolygon during a step. This is known as dynamical walking.

One of the major breakthroughs for biped walking was the introduction of the stability criterion by Vukobratovic [VB04], better known as the Zero-Moment-Point (ZMP). Several control strategies were developed and successfully applied for full sized humanoids like the mentioned HRP-2 and ASIMO. The classical control approaches demand an accurate model of the robot dynamics and use the joint position as the control variable. Systems with more recent controller are torque controlled robots. Torque controller allow a safer interaction of a robotic system with the environment or humans due to the introduced compliance property. One example for a torque controlled humanoid robot is TORO where the system gets also its name from.

Another important aspect of humanoid robot locomotion is the generation and execution of walking trajectories. The approach (presented later in more detail) in this thesis works with walking trajectories generated from a nonlinear optimization problem as stated in [WLO12, WLO14] and described via basic-splines. Certain conditions regarding constraints must hold for one step to fulfill the demand for cyclic motions.

## 1.2 Related Work

In [LNTC<sup>+</sup>11], the task of a ball catching with a robotarm was investigated. Motion trajectories were described by b-splines and found via non-linear optimization. To make optimal trajectories available at runtime, a generalization of the trajectories was studied with several machine learning methods. Finally, highly dynamical movement were applied in real-time.

The generation of energy optimal cyclic walking gaits for biped walking machines was studied in [WLO12, WLO14] where only the parametrization of the joint states was necessary. The developed motion planner treats complex motion constraint while minimizing for a defined cost function.

The generalization of optimal motions in real-time was also approached in [WHHAS13]. Learned trajectories were exactly reconstructed, while interpolated trajectories yielded a near-optimal execution.

### 1.3 Motivation

A motion planner was developed for TORO, a two-legged multibody system with 25 degrees of freedom (Fig. 1.1) at the institute of robotics and mechatronics, DLR. This nonlinear optimization based motion planner from [WLO12] is capable of generating global optimal walking trajectories for a given cost function. However, the computation of global optimal trajectories comes with a high computation cost and is therefore not applicable in real-time task.

The motivation is to solve this issue. Hence the task is formulated as follow: First, a data set which covers the whole task space is generated with the nonlinear optimizer. Second, a generalization of this data set needs to be done, to represent an underlying model of the data and third, to be able to make new predictions of motion trajectories in real-time.

### 1.4 Contribution

The contribution of this thesis is illustrated as the thick black box in Fig. 1.2. In summary, a pipeline for the generalization of optimal walking trajectories was implemented, where first a data set was generated offline for a task space range and a cost function and wrt given properties of a robotic system. Second, an underlying model with several machine learning methods estimated (also offline) and third, the prediction of new trajectories in online applications. The evaluation was done on the accuracy and runtime of the machine learning methods and the costvalue from the predicted trajectories and the resulting optimality loss through the constraint violations.

### 1.5 Outline

The thesis is structured as follows: After the introduction in this chapter, the problem statement is given in chapter 2 were also the generation of the data sets is stated. In chapter 3 first, an introduction to the field of machine learning is given and second, the theoretical foundation of the applied machine learning methods in this thesis. The application of the machine learning methods on the data sets will be described in chapter 4 and finally the results in chapter 5 presented. The conclusion and an outlook is given in the last chapter 6.



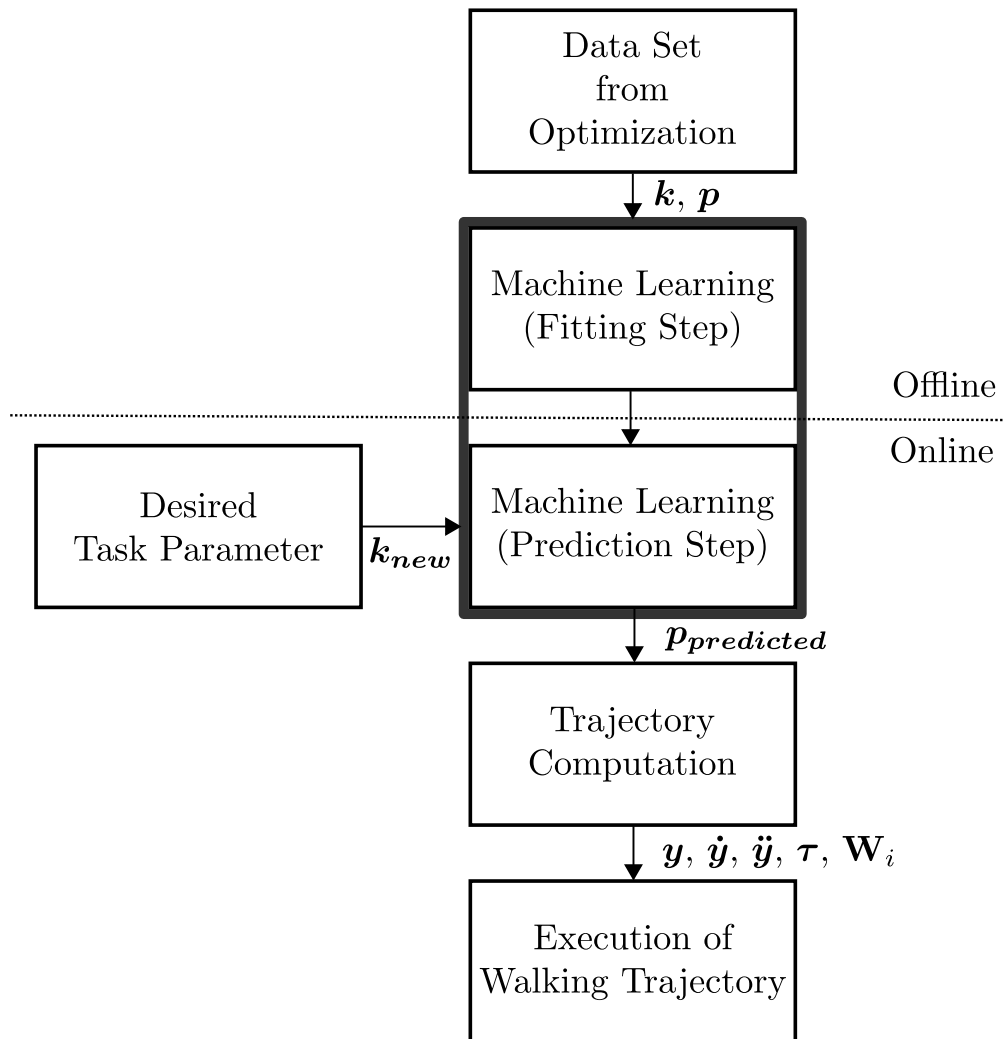


Figure 1.2: Generalization of optimal walking trajectories pipeline



---

## Chapter 2

# Problem Statement and Optimization Problem

In this chapter the problem statement of this thesis, the optimization problem and the generation of the data sets (from the optimization) which will be used in this work will be introduced. The problem statement and the definition of the optimization problem are borrowed from previous and related work at the institute of robotics and mechatronics at the DLR<sup>1</sup> [LNTC<sup>+</sup>11, WLO12, WLO14].

### 2.1 Problem Statement

The main task is to use global optimal trajectories regarding a cost function to achieve bipedal walking for a humanoid robot. The task space is specified by two task parameters, the stride-length  $k_s$  and the step time  $k_t$ . Hence,  $\mathbf{k}$  determines a full step which is divided into a single-support phase and a double support phase

$$\mathbf{k} = [k_s \quad k_t]^T \quad (2.1)$$

A trajectory which fulfills the robot kinematics and dynamics is defined in the joint space of the robot  $\mathbf{q} \in \mathbb{R}^{N_{\text{JOINTS}}}$  and described by a basic-spline (piecewise defined polynomial)  $\mathbf{q} = f_{\text{SPLINE}}(\mathbf{p}, t)$  with the parameters  $\mathbf{p} \in \mathbb{R}^{N_{\text{SPLINE}}}$ .

Knowing the joint space  $\mathbf{q}$  allows the computation of the base state  $\mathbf{x}$  through the robot kinematics and therefore the description of the full system state  $\mathbf{y}$ , see Eq. 2.2 and Eq. 2.3.

$$\mathbf{y} = [\mathbf{x} \quad \mathbf{q}]^T \quad (2.2)$$

$$\mathbf{x} = f_{\text{KIN}}^{-1}(\mathbf{q}) \quad (2.3)$$

---

<sup>1</sup>[www.dlr.de/rmc/](http://www.dlr.de/rmc/)

The full system state is used in the equation of motion of the system, as stated in Eq. 2.4 and applied in [WLO14] with the required torque  $\boldsymbol{\tau}$  which must satisfy the robot hardware limitations and the resulting contact forces  $\mathbf{W}_i$  needed for a stable contact

$$\mathbf{M}(\mathbf{y})\ddot{\mathbf{y}} + \mathbf{C}(\mathbf{y}, \dot{\mathbf{y}})\dot{\mathbf{y}} + \mathbf{g}(\mathbf{y}) = \mathbf{S}^T \boldsymbol{\tau} + \sum_{i=0}^{N_C} \mathbf{J}_i^T(\mathbf{y}) \mathbf{W}_i \quad (2.4)$$

An optimal trajectory was received from the solution of the optimization problem described below for one of the cost function defined in Eq. 2.5a a velocity cost function or Eq. 2.5b a torque cost function.

$$\Gamma_{\dot{\mathbf{q}}}(\mathbf{p}) = \int \dot{\mathbf{q}}^T \cdot \dot{\mathbf{q}} dt \quad (2.5a)$$

$$\Gamma_{\boldsymbol{\tau}}(\mathbf{p}) = \int \boldsymbol{\tau}^T \cdot \boldsymbol{\tau} dt \quad (2.5b)$$

## 2.2 Optimization Problem

The optimization problem is formulated as a minimization of a previously chosen cost function wrt equality constraints  $\mathbf{e}(\mathbf{p}, \mathbf{k})$  and inequality constraints  $\mathbf{h}(\mathbf{p})$ , see Eq. 2.7.

$$\underset{\mathbf{p}}{\text{minimize}} \quad \Gamma(\mathbf{p}) \quad (2.6)$$

$$\begin{aligned} \mathbf{e}(\mathbf{p}, \mathbf{k}) &= 0 \\ \mathbf{h}(\mathbf{p}) &< 0 \end{aligned} \quad (2.7)$$

Examples for constraints are: collision free trajectories, cyclic walking, contact forces, and joint limits for position, velocity and torques. For collision free trajectories, the safe distance between the swing foot and the surroundings is enforced. The constraint for cyclic walking ensures continued walking for the same or adjusted conditions in the next step by mirroring the step trajectories for the other foot. The friction cone and *Zero-Moment Point* (ZMP) conditions must be fulfilled by the contact force constraints to always provide a full contact to the ground. Inequality constraints in the joint position  $\mathbf{q}$  and velocity  $\dot{\mathbf{q}}$  are linear in  $\mathbf{p}$ , while constraints for joint torques  $\boldsymbol{\tau}$  and contact forces  $\mathbf{W}_i$  are highly non-linear in  $\mathbf{p}$ . An approximation of the optimization problem is achieved by computing the cost function and satisfying the given constraints at discrete timestamps for the full walking trajectory. This optimization problem was solved in previous work [LNTC<sup>+</sup>11, WLO12, WLO14].

## 2.3 Data Set Generation

Multiple machine learning methods were applied to learn the mapping of  $\mathbf{k} \rightarrow \mathbf{p}$  from the data sets computed in the optimization for the two mentioned cost functions.

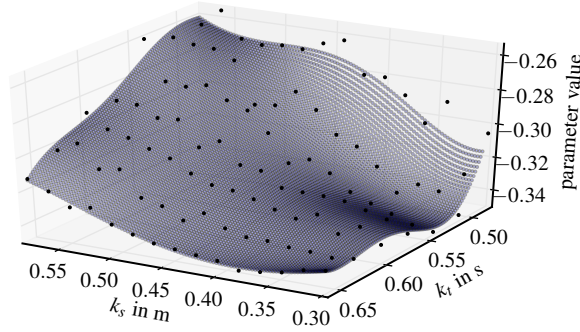


Figure 2.1: Illustration of the non-linearity of a parameter for the torque cost function. The sparse black dots are results of the optimization for given task space set, while the blue dense grid represents the predicted values from Gaussian process regression.

Generating the data sets from the optimization was done by first determining the range for the task space parameters. The stride-length  $k_s$  was varied from 0.0m to 1.2m and the step-time  $k_t$  from 0.5s to 1.08s. Second, for every pair of the task space parameters the optimization problem was solved and the trajectory parameter  $\mathbf{p}$  received. An example of such a parameter  $p_i$  is shown in Fig. 2.1. This allows to generate cyclic gaits within the specified ranges for the task parameter.

Depending on the cost function, a smooth data set in the trajectory parameter space could be generated for the velocity cost function, while noisier data sets (Fig. 2.1) in the trajectory parameter space were computed for the torque cost data set. The results in the later are twofold: First, the cost function regarding the torque  $\boldsymbol{\tau}$  is complexer than the one for the velocity  $\dot{\mathbf{q}}$  and second, since the computation takes longer, the optimizer stops earlier for certain task space parameters due to a slow convergence.

The 2D biped walker used as a model has  $N_{\text{JOINTS}} = 6$  with resulting trajectory parameter  $N_{\text{SPLINE}} = 192$ . The finally used data sets had a different amount of samples due to different noisy samples on the boundaries of the task space which were dropped for the machine learning application. One sample of the training and test set has therefore 2 task parameter  $k_s$  and  $k_t$  and 192 trajectory parameters. After learning to generalize the trajectory parameters, predicting the trajectory parameters  $\mathbf{p}_{\text{predicted}}$  for a new task parameter  $\mathbf{k}_{\text{new}}$  is possible. The applied machine learning methods are introduced in the following Chapter 3.



## Chapter 3

# Applied Machine Learning Methods

This chapters gives an introduction to machine learning which was the driving technology in this thesis. Furthermore, the applied machine learning methods will be introduced, the underlying mathematical foundation presented and the outcome for each method on a 2D example data set shown. Last but not least a clustering method, namely Gaussian Mixture Model will be presented since some data sets with local minimas needed to be clustered for the application of machine learning on subsets.

### 3.1 Machine Learning Introduction

Nowadays, several industries generate lots of data and rely on the processing of this information. The accessibility of high computing power and huge storage capacity makes this possible. Machine Learning is the approach to reason on data after building a representative model of it. For a certain process it is asumed that if we gather lots of samples from it, we can obtain with machine learning methods, were some of them applied in the thesis will be discussed in section 3.2, the underlying model of the process.

The observed data has often a certain degree of noise and is not complete so it is difficult to estimate the true model. However, different machine learning methods and approaches from several disciplines such as statistics in mathematics and artificial intelligence in computer science, as a few examples, were derived to address this issues.

With machine learning we can detect patterns automatically and based on this predict needed information or make a decision with uncertainty. Successful commercial applications such as speech recognition, computer vision, search engine page ranking and the continuous growth in further research areas like the field of robotics, made machine learning now seen as an own discipline.

Another important aspect of machine learning is the generalization of the model, since it is of high interest that the trained model provides good performance (low error rate) not only on seen data, but also on new data in the domain.

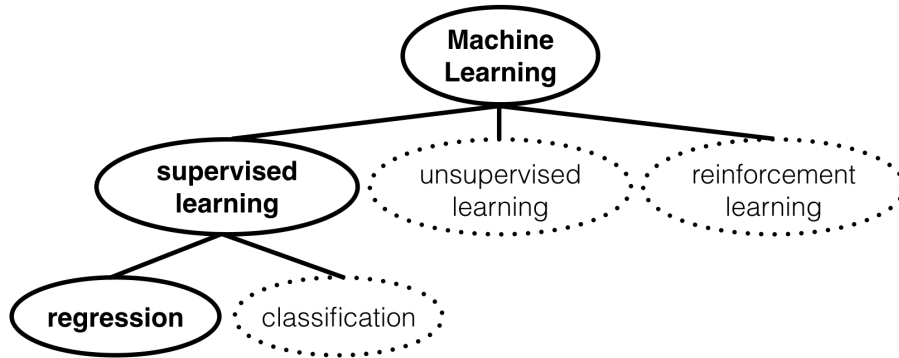


Figure 3.1: Machine Learning Approaches and Categories

The knowledge and the basic vocabulary of machine learning as stated in [Mit97, HTFF05, Mur12, JWHT13, Alp14] will be discussed below.

**Machine Learning Categories** In Fig.3.1 the main approaches of machine learning are shown (supervised, unsupervised and reinforcement learning).

Supervised learning deals with  $N$  input and output pairs:

$$\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

In the literature it is common to denote the input as  $X$ , which is in our case the task parameter  $\mathbf{k}$  and the output is denoted as  $Y$ , which are our spline parameter  $\mathbf{p}$ . Thus supervised learning provides a mapping between an input space and an output space. Furthermore, supervised learning is again divided into two subfield which are namely regression and classification. The former has a continuous output which is also referenced as quantitative, while the later deals with  $g$  groups. In this thesis supervised learning and the regression case was applied.

In unsupervised learning, data without a label, hence only input data is provided. Therefore, the main focus in this approach is to find clusters and group similar data. Between supervised and unsupervised learning is semi-supervised learning, where input data is only partially labeled so both approaches are combined to estimate a model.

The last main machine learning category is reinforcement learning. Here, an algorithm learn the output through trial and error. The goal is to find a policy upon a task can be achieved. This method is popular in the field of robotics, but was not part of this thesis.



**Data Set Handling** As discussed, we operate with machine learning on a dataset with  $N$  samples. Depending on the task and data distribution, a sufficient high sample number is desirable to guarantee a well covered taskspace.

To determine if there is a representative amount of data points the performance of the estimated model can be analysed. This consideration is due to a possible underfitting of the model.

To also overcome the opposite of this, namely overfitting, we are interested in a generalization of our model and not in a perfect fit of the model to the data, since overfitting results in a high error on new unseen data.

Therefore it is common to divide the data set beforehand into three groups, a training set, a validation set and a evaluation set.

On the training set we will fit our model and estimate the test error for the model parameter with the validation set. Upon this, we can finally assess the generalization error for the final model parameters with the evaluation set. The evaluation set is not part of the model parameter estimation. It serves only for the evaluation of the accuracy of the estimated model on not before seen data.

**Cross-Validation** Another approach to investigate over- or underfitting and in general to improve the model accuracy is to apply cross-validation in the model estimation. In cross-validation we divide as previously mentioned the data set into a small evaluation set, while dividing the rest of the samples into  $f$ -folds. Now, the model parameters are estimated for  $f - 1$ -folds, while the remaining fold is used as an validation set. In each of the  $f$  iteration, the training set and the validation set changes and we can ensure that we cover the full available sample space. Otherwise if we choose only one set to validate, we run the risk to pick a non-representative subset and hence perform worse on the evaluation set.

**Accuracy** To measure the estimated model accuracy, we can choose several options, but since we deal with a regression problem in this work, it is convenient to apply the mean squared error (MSE) see Eq. 3.1, the mean absolute error (MAE) see Eq. 3.2 or the  $R^2$ -Score in Eq. 3.3 as stated in [JWHT13]. A good model estimation results in a low MSE or low MAE, while the value of the  $R^2$ -Score should be close to 1 for a good model estimate. Furthermore, the MSE can also be normalized by the variance of the output, which serves a more meaningful interpretation.

$$MSE = \frac{1}{n} \sum_{i=1}^N (y_i - f(x_i))^2 \quad (3.1)$$

$$MAE = \frac{1}{n} \sum_{i=1}^N |y_i - f(x_i)| \quad (3.2)$$

$$R^2 \equiv 1 - \frac{\sum_i (y_i - f(x_i))^2}{\sum_i (y_i - \bar{y})^2} \quad (3.3)$$

**Parametric and non-parametric approaches** The applied machine learning methods as later discussed in 3.2 can be divided into two groups: the parametric approach and the non-parametric approach.

The parametric approach as stated in [JWHT13] is composed of two steps. First, to assume a certain underlying hyperplane shape of the given data points and second, to determine the parameters for this hyperplane. Thus we reduce with the parametric approach the problem of estimating the true underlying hyperplane to just an estimation of our model parameters, which is also the main advantage of parametric approaches, since the amount of model parameters is far lower than the amount of sample data points. However, assuming an underlying hyperplane shape claims some knowledge about it. Linear and polynomial regression is an example for a parametric approach.

While the non-parametric approach as stated in [JWHT13] can be seen as a data driven approach. In this approach, we do not make an assumption about the true underlying hyperplane but try to fit the model close to the data. Through this, we avoid the assumption of a underlying hyperplane, with the drawback of the incorporation of many samples for an precise model estimation. Examples for a non-parametric approach is the  $n$ -nearest neighbors method or the gaussian process regression.

**Example Data Set** In order to demonstrate the different outcome of the applied machine learning methods, an example data set in 2D was generated. Thus, the methods outcome can also be visually compared and the underlying methodology for each method is presented. Figure 3.2 shows the example data set, where the dash-dotted red line is the true underlying function, while the black dots are the generated noisy observations.

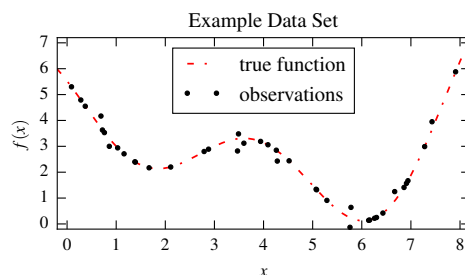


Figure 3.2: Plot of the example data set

## 3.2 Machine Learning Methods

After the introduction to the general concept of machine learning, this section presents in this study applied machine learning methods. Following methods were applied for regression in supervised learning:

- $n$ -Nearest Neighbors
- Regression Tree
- Linear and Polynomial Regression
- Support Vector Machine for Regression
- Gaussian Process for Regression

### 3.2.1 N-Nearest Neighbors

The  $n$ -nearest neighbors ( $n$ -NN) method [HTFF05, CH67] is a memory based or data driven machine learning method and hence no model estimation is required. This method operates directly on the available data set.

For the regression case, the  $n$ -NN prediction can be obtained through several approaches. These approaches are either local interpolation, local regression, averaging or local weighted averaging between  $n$  samples.

First, all distances (Euclidean see equation 3.4 or Manhattan see equation 3.5) between a new input  $\mathbf{x}_{new}$  and the available data points  $\mathbf{X}$  from the data set are computed and second, the  $n$  minimum distances are chosen.

$$d(\mathbf{x}_i, \mathbf{x}_{new}) = \sqrt{\sum_j (x_{i,j} - x_{new,j})^2} \quad (3.4)$$

$$d(\mathbf{x}_i, \mathbf{x}_{new}) = \sum_j |x_{i,j} - x_{new,j}| \quad (3.5)$$

Furthermore, a weight function is defined, where we can choose either uniform weighting see 3.6a, or a inverse distance weight function see 3.6b of each  $n$ -NN.

$$w_i = \begin{cases} 1 & (3.6a) \\ \frac{\|\mathbf{x}_{new} - \mathbf{x}_i\|^{-2}}{\sum_j \|\mathbf{x}_{new} - \mathbf{x}_j\|^{-2}} & (3.6b) \end{cases}$$

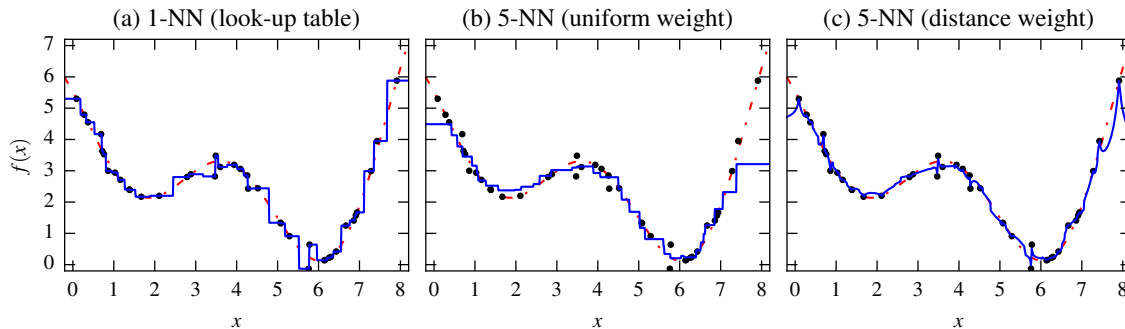


Figure 3.3:  $n$ -Nearest Neighbor Regression plots with a dash-dotted red line as a true underlying function, the black dots are the noisy observations and the blue lines as the outcome for  $n$ -NN. In (a)  $n = 1$  which results in a simple look-up table, (b) with  $n = 5$  and an uniform weight function for the  $n$ -Nearest Neighbor and (c) with  $n = 5$  and a distance weight function.

The computation of the in our case local averaged output value  $n$ -NN( $\mathbf{x}_{new}$ ) for a new input  $\mathbf{x}_{new}$  is shown in equation 3.7. Here  $n$  is the number of nearest neighbors,  $w_i$  is the weight function value between point  $\mathbf{x}_i$  and the new input  $\mathbf{x}_{new}$  and  $y_i$  is the output for point  $\mathbf{x}_i$ .

$$n\text{-NN}(\mathbf{x}_{new}) = \frac{1}{n} \sum_{i=1}^n w_i \cdot y_i \quad (3.7)$$

$N$ -nearest neighbor was applied for different  $n$  and different weight functions in this study to determine the best possible neighbor number and which weight function performance better. The result are presented in chapter 4.

In figure 3.3 the  $n$ -NN was applied on the generated 2D example data set. The first plot (a) shows the results of 1-NN regression, which is seen as a look-up table, since for a new input  $\mathbf{x}_{new}$  the output of the next possible neighbor is taken. The result for the full range is the blue line, where the occurrence of big jumps of the outcome between two close samples is high. A smoother result for the output can be achieved by increasing the number of neighbors, which is in the subplots (b) and (c) 5. While in subplot (b) a uniform weight function was applied, subplot (c) displays the outcome of a inverse distance weight function. Uniform weighting generalizes better in the sense of fewer big changes in the output value for two close samples. Prediction close to the border of the data set range are worse than those from within, since on the border, points for the local weighted averaging are taken from one side of the sample.

### 3.2.2 Regression Tree

An additional conceptual simple and fast regression method is the regression tree (RT) [HTFF05]. The RT is also a non-parametric approach, where the whole task space is partitioned into  $M$  regions  $\mathbf{R}_m$  and the outcome is the average of the samples from this leaf  $m$ . The minimum leaf size or samples per region was one sample. A RT has a root node and at least two branches as illustrated in figure 3.4. Inequality conditions in each node can be composed of more complex inequality conditions than the one in figure 3.4.

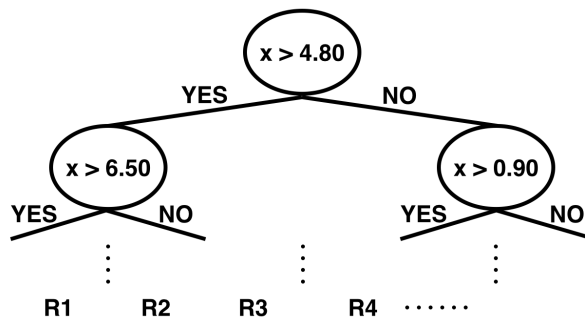


Figure 3.4: Regression Tree

In our task space we have  $N$  observations, where each input  $\mathbf{x}_i$  has a corresponding output  $y_i$ . Starting from the root node, the RT partitions the task space until a specified maximum depth  $d$ . To simplify the splitting, the RT is restricted to perform only binary splits. Thus, we first split the task space into two regions, then each region again into two subregions and so on. This procedure is performed until some criterion is achieved.

A RT predicts  $y_m$  for a region  $\mathbf{R}_m$  as  $c_m$ , which is a constant value in the corresponding region, see equation 3.8 as stated in [HTFF05]. The constant is for example the average of samples in each region (Equation 3.9).

$$\hat{f}(\mathbf{x}_i) = \sum_{m=1}^M c_m I\{\mathbf{x}_i \in \mathbf{R}_m\} \quad (3.8)$$

$$c_m = \text{avg}(y_i | \mathbf{x}_i \in \mathbf{R}_m) \quad (3.9)$$

Finding the best splitting point is challenging. [HTFF05] suggest a greedy approach, where they define two regions  $\mathbf{R}_{\text{left}}$  and  $\mathbf{R}_{\text{right}}$  (Equation 3.10), determined by the splitting variable  $j$  and the splitting point  $s$ .

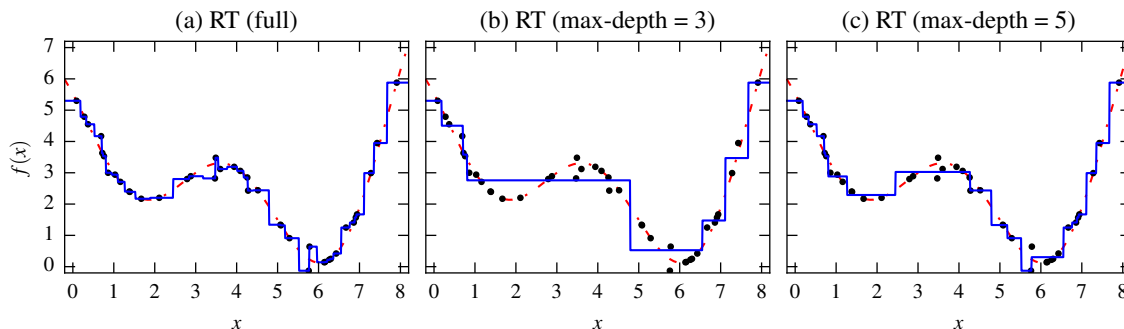


Figure 3.5: Applied Regression Tree plots with a dash-dotted red line as a true underlying function, the black dots are the noisy observations and the blue lines as the outcome for the RT. In (a) with no limited max-depth of the tree which also results in a simple look-up table, (b) with a set maximum tree depth of 3 and (c) with a maximal tree depth of 5.

$$\mathbf{R}_{left}(j, s) = \{\mathbf{x} | \mathbf{x}_j \leq s\} \quad \text{and} \quad \mathbf{R}_{right}(j, s) = \{\mathbf{x} | \mathbf{x}_j > s\} \quad (3.10)$$

The two introduced variable are estimated by minimizing equation 3.11 with 3.12. Applying this on all samples in the task space is faster than a minimization of the mean-squared error in each region to determine a splitting variable and a splitting point.

$$\min_{j,s} [\min_{c_{left}} \sum_{\mathbf{x}_i \in \mathbf{R}_{left}} (y_i - c_{left})^2 + \min_{c_{right}} \sum_{\mathbf{x}_i \in \mathbf{R}_{right}} (y_i - c_{right})^2] \quad (3.11)$$

$$c_{left} = avg(y_i | \mathbf{x}_i \in \mathbf{R}_{left}(j, s)) \quad \text{and} \quad c_{right} = avg(y_i | \mathbf{x}_i \in \mathbf{R}_{right}(j, s)) \quad (3.12)$$

The result of a RT is local averaging of the task space and the partitioned task space can be described by one fitted RT. Without limiting the RT depth, the estimated model overfits as illustrated in figure 3.5 (a). The result is similar to 1-NN in the previous section, a look-up table since each leaf or region has only one sample and there are as many leaves as observations in the task space. Figure 3.5 (b) shows a RT with a depth of 3, while figure 3.5 (c) a RT with a depth of 5, which performs better with respect of the error for new unseen data. To improve the generalization of RT, trees can be pruned to reduce the amount of splits and therefore to reduce the depth. Regions with small changes in their average value can be combined to one region.

### 3.2.3 Linear and Polynomial Regression

A parametric approach in this study is the linear and polynomial regression [HTFF05]. With this machine learning methods the relation between the input  $X$  and the Output  $Y$  is assumed to be either linear, or this mapping is done with a  $n$ -th degree polynomial. Therefore we approximate a true underlying function  $f(\mathbf{x})$  of the observed noisy data (Eq. 3.13) with  $\hat{f}(\mathbf{x})$  (Eq. 3.14). For  $x_0 = 1$ , the equation can be rewritten as in Eq. 3.15. The random noise is assumed to be normally distributed with zero mean. However, polynomial regression is not considered to be non-linear. Although it fits non-linear hyperplanes, the free model parameters change linearly.

$$y_i = f(\mathbf{x}_i) + \epsilon_i \quad (3.13)$$

$$\hat{f}(\mathbf{x}_i) = \omega_0 + \sum_{j=1}^n \omega_j x_{i,j} = \sum_{j=0}^n \omega_j x_{i,j} \quad (3.14)$$

$$\hat{f}(\mathbf{x}_i) = \boldsymbol{\omega}^T \mathbf{x}_i \quad (3.15)$$

The extension of linear regression to polynomial regression can be achieved by the construction of polynomial features  $\mathbf{z}_i$  from the input  $\mathbf{x}_i$  as in Eq. 3.16. The resulting model (Eq. 3.17) is similar to Eq. 3.15 since it is also a linear model (linear in the parameter). Applying polynomial features comes at a cost. The computational complexity increases as the number of free model parameter increases and complex models tend to overfitting. This must be considered in the application of the method.

$$\mathbf{z}_i = \{1, \mathbf{x}_i, \mathbf{x}_i^2, \mathbf{x}_i^3, \dots, \mathbf{x}_i^n\} \quad (3.16)$$

$$\hat{f}(\mathbf{z}_i) = \boldsymbol{\omega}^T \mathbf{z}_i \quad (3.17)$$

With the residual sum of squares (Eq. 3.18) we can compute the free model parameter  $\boldsymbol{\omega}$  (Eq. 3.21), where  $X$  are all input samples and  $Y$  all output samples.

$$\text{RSS}(\boldsymbol{\omega}) = \sum_{i=1}^N (y_i - \hat{f}_{\boldsymbol{\omega}}(\mathbf{x}_i))^2 \quad \text{and} \quad \text{RSS}(\boldsymbol{\omega}) = \sum_{i=1}^N (y_i - \hat{f}_{\boldsymbol{\omega}}(\mathbf{z}_i))^2 \quad (3.18)$$

$$\boldsymbol{\omega} = (X^T X)^{-1} X^T Y \quad (3.19)$$

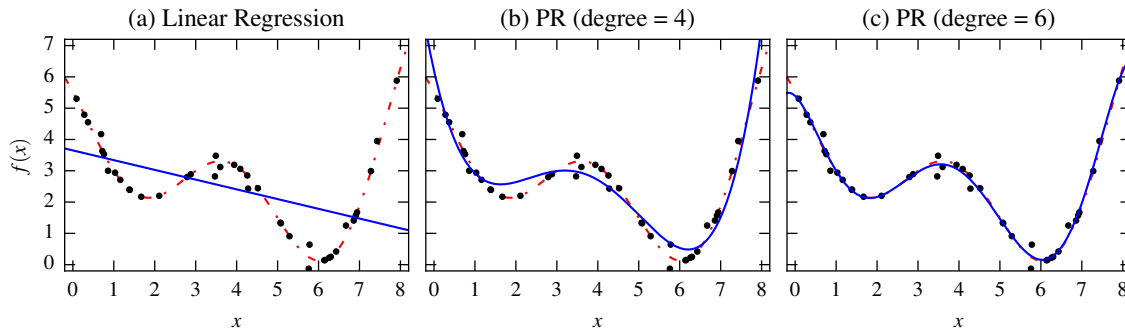


Figure 3.6: Linear / Polynomial Regression plots with a dash-dotted red line as a true underlying function, the black dots are the noisy observations and the blue lines as the outcome of the regression. In (a) Linear Regression, (b) Polynomial Regression with a fit of a 4th order polynomial and (c) with a fit of a 6th order polynomial.

As mentioned, the increase of the polynomial order runs the risk to overfit the data. Furthermore, the free model parameter can become really big. To prevent this we can apply ridge regression [HTFF05] which shrinks the model parameter by a penalizing coefficient  $\lambda$  (Eq. 3.20).

$$\text{RSS}(\omega, \lambda) = \sum_{i=1}^N (y_i - \hat{f}_\omega(\mathbf{z}_i))^2 + \lambda \omega^T \omega \quad (3.20)$$

$$\omega_{\text{ridge}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{Y} \quad (3.21)$$

The results on the example data set for linear and polynomial regression are illustrated in Fig. 3.6. Linear regression should always be considered since it is a good initial guess of a underlying function. However, as seen in (a) the fitted model poses a bad estimate on the example data set. Extending linear regression to polynomial features results in a better fit. In the subplot (b) a 4th order polynomial was applied which is a better guess than before but poses still a recognizable model error, while subplot (c) shows the best result for polynomial regression with a 6th order polynomial. Ridge Regression was also applied in (c) to shrink the free model parameters.



### 3.2.4 Support Vector Regression

One of the more sophisticated machine learning methods applied in this thesis is the support vector machine (SVM). The SVM was initially developed for classification tasks but was then extended so serve in regression problems where also multidimensional hyperplanes can be fitted to the given data set  $(\mathbf{x}_i, y_i), i = 1, \dots, N$ . The term support vector regression (SVR) as introduced in [VGS97, DBK<sup>+</sup>97, SS04] is used for this method.

In contrast to the method (linear regression) from the previous section, where all samples are taken to estimate the free model parameter at once, the SVR uses only as many samples as defined support vectors (SV) which are by far less. This approach is better known as a sparse estimation and an advantage of SVR in high dimensions.

Within the SVR method, there are two different approaches for defining the SVs. This section will start with the  $\epsilon$ -SVR, where the free parameters are  $\epsilon$  and the regularization  $C$ . A second approach, namely the  $\nu$ -SVR introduced in [SSWB00, CL02], with  $\nu$  and  $C$  as the free parameters will be investigated in this study. Additionally, the kernel trick will be introduced and the application of both SVR approaches on the example data set shown.

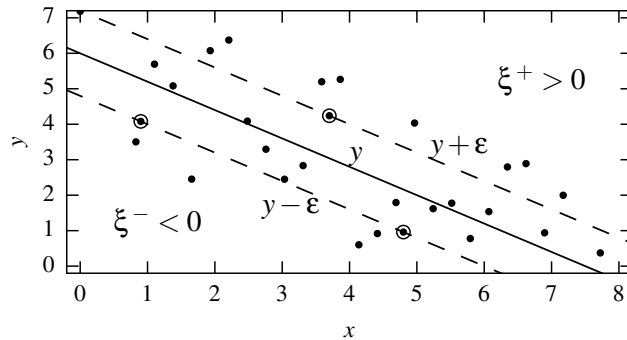


Figure 3.7: From [SS04],  $\epsilon$ -tube and slack variable  $\xi$

**$\epsilon$ -SVR** As with the previous method we want to estimate the underlying function for a given data set, see Eq. 3.22. A good estimate is achieved by minimizing the risk function in Eq. 3.23 with which the training error and the model complexity can be controlled. The in [VGS97] defined  $\epsilon$ -insensitive loss function in Eq. 3.24 penalizes all samples outside of the  $\epsilon$ -tube. From [SS04], see Fig. 3.7, we can say that  $\epsilon$ -SVR does not care about errors with a lower deviation than  $\epsilon$  from the estimated model.

$$y_i = f(\mathbf{x}_i, \boldsymbol{\omega}) = b + \sum_{j=1}^M \omega_j \phi_j(\mathbf{x}_i) = b + \boldsymbol{\omega}^T \boldsymbol{\phi}(\mathbf{x}_i) \quad (3.22)$$

$$R(y, \hat{f}(\mathbf{x}_i), \boldsymbol{\omega}) = C \frac{1}{l} \sum_{i=1}^l |y_i - \hat{f}_{\boldsymbol{\omega}}(\mathbf{x}_i)|_{\epsilon} + \frac{1}{2} \|\boldsymbol{\omega}\|^2 \quad (3.23)$$

$$|y_i - \hat{f}_{\boldsymbol{\omega}}(\mathbf{x}_i)|_{\epsilon} = \begin{cases} 0 & \text{if } |y_i - \hat{f}(\mathbf{x}_i, \boldsymbol{\omega})| < \epsilon, \\ |y_i - \hat{f}(\mathbf{x}_i, \boldsymbol{\omega})| - \epsilon & \text{otherwise} \end{cases} \quad (3.24)$$

Furthermore, a soft-margin and hence a slack variable  $\xi$  (Eq. 3.26) is introduced [SS04] which describes the degree and direction of the deviation of a sample from the  $\epsilon$ -tube, where  $\xi_i^+$  penalizes samples above the  $\epsilon$ -tube and  $\xi_i^-$  those below. Again, free model parameter can be estimated by minimizing the updated risk function in Eq. 3.25 which is subject to the constraints in Eq. 3.26.

$$R(\xi_i^+, \xi_i^-, \boldsymbol{\omega}) = C \frac{1}{l} \sum_{i=1}^l (\xi_i^+ + \xi_i^-) + \frac{1}{2} \|\boldsymbol{\omega}\|^2 \quad (3.25)$$

$$y_i - (\boldsymbol{\omega}^T \phi(\mathbf{x}_i) + b) \leq \epsilon + \xi_i^+ \quad (3.26a)$$

$$(\boldsymbol{\omega}^T \phi(\mathbf{x}_i) + b) - y_i \leq \epsilon + \xi_i^- \quad (3.26b)$$

$$\xi_i^+, \xi_i^- \geq 0, i = 1, \dots, l \quad \epsilon \geq 0$$

The risk function minimization is achieved by the application of the Lagrange multiplier technique (Eq. 3.27). Therefore partial derivatives wrt.  $\boldsymbol{\omega}, b, \xi_i^+, \xi_i^-$  are computed and set to zero as in Eq. 3.28. Now, we can insert those results in our function estimation from Eq. 3.22 and we receive Eq. 3.29. The coefficients  $\alpha$  can be obtained through quadratic (convex) programming. The amount of support vector can be determined by finding the indices  $i$  where  $\xi_i^+ = 0$  or  $\xi_i^- = 0$  and for  $\alpha$  where following condition holds:  $0 < \alpha < C$

$$\begin{aligned} L = & C \frac{1}{l} \sum_{i=1}^l (\xi_i^+ + \xi_i^-) + \frac{1}{2} \|\boldsymbol{\omega}\|^2 - \sum_{i=1}^l (\mu_i^+ \xi_i^+ + \mu_i^- \xi_i^-) \\ & - \sum_{i=1}^l \alpha_i^+ (\epsilon + \xi_i^+ + y_i - (\boldsymbol{\omega}^T \phi(\mathbf{x}_i) + b)) \\ & - \sum_{i=1}^l \alpha_i^- (\epsilon + \xi_i^- - y_i + (\boldsymbol{\omega}^T \phi(\mathbf{x}_i) + b)) \end{aligned} \quad (3.27)$$

$$\frac{\partial L}{\partial \boldsymbol{\omega}} = 0 \Rightarrow \hat{\boldsymbol{\omega}} = \sum_{i=1}^l (\alpha_i^+ - \alpha_i^-) \phi(\mathbf{x}_i) \quad (3.28a)$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^l (\alpha_i^+ - \alpha_i^-) = 0 \quad (3.28b)$$

$$\frac{\partial L}{\partial \xi_i^+} = 0 \Rightarrow C = l(\alpha_i^+ + \mu_i^+) \quad (3.28c)$$

$$\frac{\partial L}{\partial \xi_i^-} = 0 \Rightarrow C = l(\alpha_i^- + \mu_i^-) \quad (3.28d)$$

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^l (\alpha_i^+ - \alpha_i^-) \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b \quad (3.29)$$

**$\nu$ -SVR** The other SVR approach is introduced in [SSWB00] as the  $\nu$ -SVR. Although  $\nu$ -SVR is basically the same as the previously mentioned  $\epsilon$ -SVR, it omits the determination of a desired accuracy beforehand through the  $\epsilon$ -parameter. Instead it replaces the free parameter  $\epsilon$  with the  $\nu$ -parameter, which allows an automatical minimization of the accuracy  $\epsilon$ . The risk function and the Lagrangian are extended as seen in Eq. 3.30 and Eq. 3.31, and the solution can be found through the same procedure as before for Eq. 3.29.

$$R(\xi_i^+, \xi_i^-, \boldsymbol{\omega}, \epsilon) = C(\nu\epsilon + \frac{1}{l} \sum_{i=1}^l (\xi_i^+ + \xi_i^-)) + \frac{1}{2} \|\boldsymbol{\omega}\|^2 \quad (3.30)$$

$$\begin{aligned} L = C\nu\epsilon + C\frac{1}{l} \sum_{i=1}^l (\xi_i^+ + \xi_i^-) + \frac{1}{2} \|\boldsymbol{\omega}\|^2 - \sum_{i=1}^l (\mu_i^+ \xi_i^+ + \mu_i^- \xi_i^-) \\ - \sum_{i=1}^l \alpha_i^+ (\epsilon + \xi_i^+ + y_i - (\boldsymbol{\omega}^T \phi(\mathbf{x}_i) + b)) \\ - \sum_{i=1}^l \alpha_i^- (\epsilon + \xi_i^- - y_i + (\boldsymbol{\omega}^T \phi(\mathbf{x}_i) + b)) \end{aligned} \quad (3.31)$$

The parameters of the former SVR,  $C$  and  $\epsilon$  are in the range  $[0, \infty)$  while  $\nu$  in  $\nu$ -SVR is in the range  $[0, 1)$ . With  $\nu$ -SVR, a more meaningful representation of the penalty parameter is achieved, since the  $\nu$ -parameter represents an upper bound on the fraction of training samples and a lower bound on the fraction of samples which are support vectors. For both SVR methods, the free parameters are found via a grid-search.

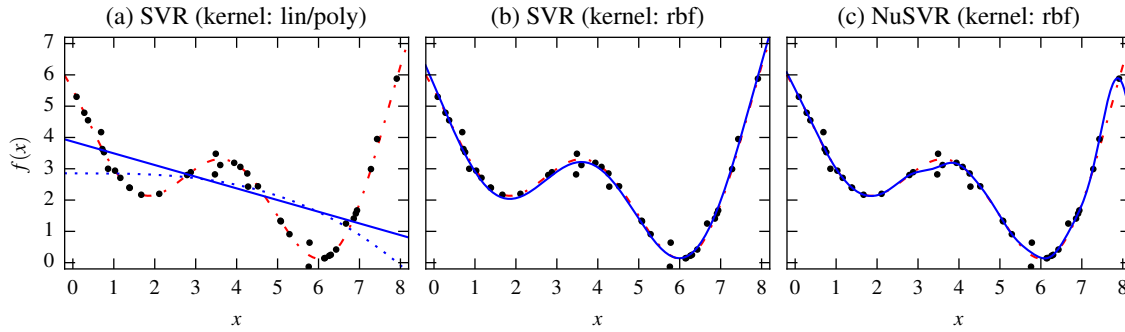


Figure 3.8: Support Vector Regression plots with a dash-dotted red line as a true underlying function, the black dots are the noisy observations and the blue (full/dotted) lines as the outcome of the regression. In (a) applied  $\epsilon$ -SVR with a linear kernel (blue full line) and with a polynomial kernel (blue dotted line), in (b) a  $\epsilon$ -SVR with a rbf kernel and in (c)  $\nu$ -SVR also with a rbf kernel.

**Kernel Trick** Applying kernels means transforming not linear fitable data into a higher dimension where we can determine the multidimensional hyperplane. Therefore, a linear kernel (Eq. 3.32a), a polynomial kernel (Eq. 3.32b) and a gaussian kernel (Eq. 3.32c) as stated below were applied in this study:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j) \quad (3.32a)$$

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + c)^n \quad (3.32b)$$

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{(\mathbf{x}_i - \mathbf{x}_j)^2}{2\sigma}\right) \quad (3.32c)$$

Kernelizing the estimated function from Eq. 3.29 leads to the solution of the estimated function as shown in Eq. 3.33.

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^l (\alpha_i^+ - \alpha_i^-) \kappa(\phi(\mathbf{x}_i), \phi(\mathbf{x})) + b \quad (3.33)$$

**Application of SVR** As with the previous methods, the  $\epsilon$ -SVR and  $\nu$ -SVR was applied on the example data set. In Fig. 3.8 (a)  $\epsilon$ -SVR was applied with a linear kernel (blue full line) and with a 4th order polynomial kernel (blue dotted line). The function estimation failed for both kernels. In the subplot (b),  $\epsilon$ -SVR was applied with a gaussian kernel and yielded a good estimation with a small error, while in subplot (c)  $\nu$ -SVR, also with a gaussian kernel was applied which performed similar good, however slightly different, e.g. for  $x$  between 3 and 4.

### 3.2.5 Gaussian Process Regression

Gaussian Process Regression (GPR) [RW06, Ebd08] is considered to be like the SVR from the previous section a kernel machine. While the SVR is sparse and therefore fast, it lacks of a probabilistic output. That's were the GPR comes in: With GPR we define a distribution over functions. Starting from a prior, where we incorporate our initial belief over functions, we compute a posterior with the help of Bayes' rule and marginal likelihood after some observations. The key idea is that for similar samples  $\mathbf{x}$  and  $\mathbf{x}'$ , the output of the functions of  $f(\mathbf{x})$  and  $f(\mathbf{x}')$  are expected to be similar.

Again, we use the standard linear regression model with Gaussian noise  $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ , see Eq. (3.34). The GP can be written as in Eq. (3.35), where the Kronecker delta  $\delta(\mathbf{x}, \mathbf{x}')$  is 1 iff  $\mathbf{x} = \mathbf{x}'$  and otherwise 0, with mean  $m(\mathbf{x})$  and covariance function or kernel  $\kappa(\mathbf{x}, \mathbf{x}')$  (Eq. (3.36)).

$$\mathbf{y} = f(\mathbf{x}) + \epsilon \quad (3.34)$$

$$\mathbf{y} \sim \mathcal{GP}(m(\mathbf{x}), \kappa(\mathbf{x}, \mathbf{x}') + \sigma_n^2 \delta(\mathbf{x}, \mathbf{x}')) \quad (3.35)$$

$$m(\mathbf{x}) = \mathbb{E}[\mathbf{y}] \quad (3.36a)$$

$$\kappa(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(\mathbf{y} - m(\mathbf{x}))(\mathbf{y}' - m(\mathbf{x}'))^T] \quad (3.36b)$$

A popular choice for a covariance function is the squared-exponential kernel, as stated below in Eq.(3.37), with  $\mathbf{M} = l^{-2}\mathbf{I}$ .

$$\kappa(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \mathbf{M}(\mathbf{x} - \mathbf{x}')\right) + \sigma_n^2 \delta(\mathbf{x}, \mathbf{x}') \quad (3.37)$$

After the observation of the training data set we can make prediction with the posterior for new data. Therefore, we need to compute the covariance values from the covariance function for every observation sample. Here,  $\mathbf{K}$  represents the covariance between training samples (Eq.(3.38a)),  $\mathbf{K}_*$  the training-test samples covariance (Eq.(3.38b)) and  $\mathbf{K}_{**}$  the covariance of the test sample (Eq.(3.38c)).

$$\mathbf{K} = \begin{bmatrix} \kappa(x_1, x_1) & \kappa(x_1, x_2) & \cdots & \kappa(x_1, x_n) \\ \kappa(x_2, x_1) & \kappa(x_2, x_2) & \cdots & \kappa(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(x_n, x_1) & \kappa(x_n, x_2) & \cdots & \kappa(x_n, x_n) \end{bmatrix} \quad (3.38a)$$

$$\mathbf{K}_* = \begin{bmatrix} \kappa(x_*, x_1) & \kappa(x_*, x_2) & \cdots & \kappa(x_*, x_n) \end{bmatrix} \quad (3.38b)$$

$$\mathbf{K}_{**} = \kappa(x_*, x_*) \quad (3.38c)$$

Recalling [RW06, Ebd08] we know that Eq.(3.39) holds, so we compute the conditional distribution of  $y_*$  given  $\mathbf{y}$  (Eq.(3.40)).

$$\begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix} \sim \mathcal{N} \left( \mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{K}_*^T \\ \mathbf{K}_* & \mathbf{K}_{**} \end{bmatrix} \right) \quad (3.39)$$

$$y_* | \mathbf{y} \sim \mathcal{N} (\mathbf{K}_* \mathbf{K}^{-1} \mathbf{y}, \mathbf{K}_{**} - \mathbf{K}_* \mathbf{K}^{-1} \mathbf{K}_*^T) \quad (3.40)$$

From the conditional distribution in Eq.(3.40) we receive with the mean our best estimation for a new sample  $\mathbf{x}_*$  (Eq.(3.41)), while the uncertainty of the output is described by Eq.(3.42). Furthermore, the expected value for  $y_*$  in Eq. (3.41) can be rewritten, which is similar to the solution in the previous section for SVR, with  $\boldsymbol{\alpha} = \mathbf{K}^{-1} \mathbf{y}$ .

$$\mathbb{E}(y_*) = \mathbf{K}_* \mathbf{K}^{-1} \mathbf{y} = \sum_{i=1}^n \alpha_i \kappa(x_i, x_*) \quad (3.41)$$

$$\mathbb{V}(y_*) = \mathbf{K}_{**} - \mathbf{K}_* \mathbf{K}^{-1} \mathbf{K}_*^T \quad (3.42)$$

The posterior variance in Eq.(3.42) is smaller than the prior variance from Eq.(3.38c), since a positive term is subtracted. Another interpretation is that the data brings in information, therefore the uncertainty is smaller. Furthermore, the posterior variance depends only on the input data.

**GP Hyperparameters** After we learned how to compute the posterior from the prior, it is of interest to know how to determine the hyperparameters  $\boldsymbol{\theta} = \{l, \sigma_f, \sigma_n\}$  of the GP with respect of the given data set. The hyperparameters are within the kernel function (Eq.(3.37)) and are as follows:

- characteristic length-scale  $l$
- vertical scale of functions  $\sigma_f^2$
- noise variance  $\sigma_n^2$

There are two options to determine the kernel parameters, either via an exhaustive grid search over discrete values which can be slow, or with a continuous optimization. Therefore, we need to find the best  $p(\boldsymbol{\theta} | \mathbf{x}, \mathbf{y})$ , that according to Bayes rule means the maximization of the marginal log likelihood w.r.t. hyperparameters  $\boldsymbol{\theta}$ , see Eq.(3.43).

$$L = \log p(\mathbf{y} | \mathbf{X}, \boldsymbol{\theta}) = -\frac{1}{2} \log |\mathbf{K}| - \frac{1}{2} \mathbf{y}^T \mathbf{K}^{-1} \mathbf{y} - \frac{n}{2} \log(2\pi) \quad (3.43)$$

The first term is known as a complexity penalty term, the second one is the data fit term, while the last represents a constant.

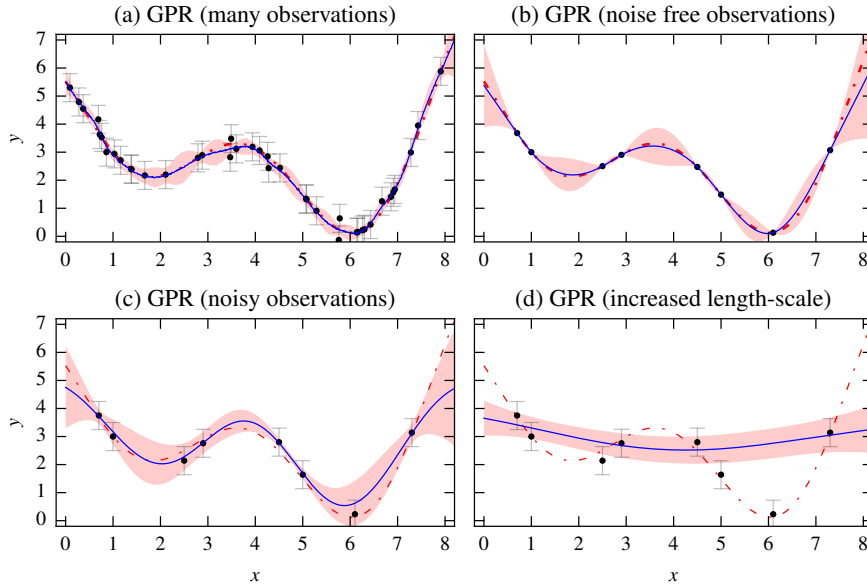


Figure 3.9: Gaussian Process Regression plots with a dash-dotted red line as a true underlying function, the black dots are the observations, the blue (full/dotted) lines as the outcome of the regression and the red filled areas as the 95% confidence interval. In (a) GPR was applied on many observations. In (b) the amount of observations was reduced, while also assuming noise free observations. In (c) and (d) the GPR was applied with noisy observations, while in (c) the estimated model fits well, the length-scale in (d) is to high.

To estimate the appropriate hyperparameters, we apply numerical optimization e.g. conjugate gradients, on the partial derivatives from the marginal log likelihood (Eq.(3.44)), with  $\boldsymbol{\alpha} = \mathbf{K}^{-1}\mathbf{y}$ :

$$\frac{\partial L}{\partial \theta_j} = \frac{\partial}{\partial \theta_j} \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \frac{1}{2} \mathbf{y}^T \mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta_j} \mathbf{K}^{-1} \mathbf{y} - \frac{1}{2} \text{tr}(\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta_j}) \quad (3.44a)$$

$$= \frac{1}{2} \text{tr} \left( (\boldsymbol{\alpha} \boldsymbol{\alpha}^T - \mathbf{K}^{-1}) \frac{\partial \mathbf{K}}{\partial \theta_j} \right) \quad (3.44b)$$

Since the GP uses all samples to make new prediction, one must be careful with the amount of observations. Especially in high dimensions with many features, the computational complexity becomes infeasible. This is one drawback of GPs. In Fig. 3.9 some examples for GPR are shown. Starting with the subplot (a) where many noisy observations were incorporated, the GPR leads to a good model fit with a low error. The GPR for noise free observation, see subplot (b), leads to a equally good model while incorporating much less samples. In addition, since noise should always be considered, (c) and (d) show the outcome of GPR for noisy data. Whereas optimal hyperparameter were applied in (c), subplot (d) shows the result for a higher length scale  $l$ .

### 3.3 Clustering with Gaussian Mixture Models

In contrast to previously introduced machine learning methods for supervised learning, clustering with Gaussian Mixture Models (GMM) [Rey09] is an unsupervised learning approach. The task is to find for a given data set  $\mathcal{D}$  the  $N$  predefined clusters. A Gaussian mixture distribution can be written as in Eq. 3.45. The mixing coefficients  $\pi_i$  are prior probabilities for every cluster and must satisfy the conditions in Eq. 3.46. Furthermore, the samples from a cluster are assumed to be normal distributed with mean  $\boldsymbol{\mu}_i$  and covariance  $\boldsymbol{\Sigma}_i$ , see Eq. 3.47.

$$p(\mathbf{x}) = \sum_{i=1}^N p(k)p(\mathbf{x}|k) = \sum_{i=1}^N \pi_i \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \quad (3.45)$$

$$\sum_{i=1}^N \pi_i = 1 \quad 0 \leq \pi_i \leq 1 \quad (3.46)$$

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}_i|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)\right) \quad (3.47)$$

The GMM can be fitted to the data by finding the parameters  $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}$ , through the maximization of the log likelihood [B<sup>+</sup>06] in Eq. 3.48. It is however not possible to find a solution in closed form due to the sum of the term inside the logarithm in Eq. 3.48.

$$\log p(\mathcal{D}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{j=1}^M \log \left[ \sum_{i=1}^N \pi_i \mathcal{N}(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \right] \quad (3.48)$$

A feasible solution for finding the maximum log likelihood wrt to the parameter is an iterative approach better known as the expectation-maximization algorithm [B<sup>+</sup>06]. In the first step, the initialization step, random prior probabilities  $\boldsymbol{\pi}$ , random means  $\boldsymbol{\mu}$  and covariance  $\boldsymbol{\Sigma}$  with non-zero entries are chosen and the log likelihood computed. After this, the posterior probability called responsibility (Eq. 3.49) is computed with the current parameter in the expectation step. This is seen as a soft assignment of every observation to a cluster.

$$r_{ik} = \frac{p(k)p(\mathbf{x}|k)}{p(\mathbf{x})} = \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{i=1}^N \pi_i \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)} \quad (3.49)$$



In the maximization step, the means (Eq. 3.50a) and the covariance (Eq. 3.50b) are reestimated with the responsibility from the previous step. Also the cluster weights (Eq. 3.50c) are updated, where  $N_k$  (Eq. 3.50d) is the number of samples in a cluster.

$$\boldsymbol{\mu}_k^{new} = \frac{1}{N_k} \sum_{i=1}^N r_{ik} \mathbf{x}_i \quad (3.50a)$$

$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{N_k} \sum_{i=1}^N r_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k^{new})(\mathbf{x}_i - \boldsymbol{\mu}_k^{new})^T \quad (3.50b)$$

$$\pi_k^{new} = \frac{N_k}{N} \quad (3.50c)$$

$$N_k = \sum_{i=1}^N r_{ik} \quad (3.50d)$$

Last but not least, the convergence of the log likelihood or of the parameter  $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}$  is evaluated. If there is still a change of the log likelihood, the expectation step (E-Step) and the maximization step (M-Step) are repeated.



## Chapter 4

# Application of Methods on Data

The previous chapter presented the theoretical foundation of the applied machine learning methods for regression problems in this thesis. With this knowledge, each method was applied on the data sets introduced in chapter 2 to find the best fit of a model of the data and subsequently a low error for new predictions.

Therefore, chapter 4.1 starts with the search for the best parametrization of the machine learning methods to fit and predict the data while still considering generalization of the models on the velocity cost data set. This chapter presents the results for every method parametrization, while the best method parameters will be discussed in the following chapter 5. The outcome for each method with the best parametrization in the task space is illustrated in chapter 4.2 for one trajectory parameter as an example. However, this was done for all trajectory parameters. In addition, chapter 4.3 shows the application of clustering with the GMM on the torque cost data set. This step was necessary since local subsets differ from neighboring subsets due to a local convergence in the optimizer for certain task space ranges and task space parameters. It is of interest if clustering and fitting of subsets resulted in lower errors.

### 4.1 Methods Parameter Determination

From chapter 3 we know that in machine learning there are parametric and non-parametric methods. The described parametrization in this chapter differ from the previous chapter, since this parameters are those, which were forwarded to *scikit-learn* [PVG<sup>+</sup>11] to call a class with this values and apply it. The amount of parameters for each method differ. Furthermore there are some parameters influencing the outcome significant more than others. Reducing the amount of possible combinations of parameters for the grid-search (explained on the next page), a few less influential parameter were kept at a constant value while varying the other parameters to compare the performance regarding the accuracy and the runtime. Although the accuracy was the driving factor for the selection of the best parametrization, the runtime of each methods was also investigated to prevent surprising outcome.

**Grid Search** For finding the best parametrization of each method regarding the data a technique called grid-search was applied. Another possible solution to this problem would be determining the parameters via numerical optimization. But since the parameters for each method are either discrete or the total range for a parameter can be restricted to certain values, grid search is a fast and simple way to find the best parametrization. Thus a possible feasible parametrization for each method was chosen and as many runs to fit and predict on the data as combinations of parameters were applied. This is illustrated in Fig. 4.1 where a grid for two possible parameters was investigated as it was the case for the SVR.

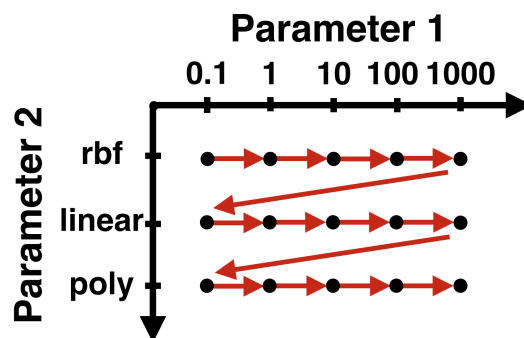


Figure 4.1: Grid search illustration for two parameters

The possible parameters of each method are introduced and the results for each parametrization is shown on the following pages. The best result is highlighted in red, while special cases are green. Additionally cross-validation with 11 folds was applied. For each method and parametrization, the average of the results from all folds was taken to determine the mean squared error (MSE) and also the runtime. This values are visualized in the plots.

**Nearest Neighbor** For the  $n$ -NN the number of neighbors for the regression was investigated which was in the range from 1 to 10 neighbors. Where 1 neighbor is a simple look-up table. Additionally, the outcome for different weight functions was compared. See Fig. 4.2 for the results for a uniform weight function and Fig. 4.3 for a distance weight function. The results for the distance weighth function are better than those for the uniform weight function, while the runtime increases by a factor of 10 due to the computation of the distance between neighbors. Example class instantiation and assignments in python are as stated below:

```
unn = NearestNeighbor(n=3, weight='uniform')
```

```
dnn = NearestNeighbor(n=5, weight='distance')
```

**Regression Tree** The RT has one main parameter – the max. depth of the tree. This parameter was varied between 1 and 9. Other parameters are the minimum leaf size which was one and the minimum samples to split which was two, but this value were kept constant. Furthermore, a RT without depth limits was applied. This results in a look-up table with as many leaves as samples. See Fig. 4.4 for the outcome. An example call can be seen below:

```
rt = DecisionTreeRegressor(max_depth=5)
```

**Polynomial Regression** In the PR, a feasible regularization factor was found and kept constant (at 0.005), while the degree of the polynomial was altered. Therefore polynomials from degree 1 to 10 were applied. The results are shown in Fig. 4.5. The linear regression (degree=1) completely fails in representing the underlying model of the data.

```
reg = make_pipeline(PolynomialFeatures(degree=5), Ridge(alpha=0.005))
```

**Support Vector Regression** For SVR there are two cases: The  $\epsilon$ -SVR and the  $\nu$ -SVR. For the former an  $\epsilon$  value of 0.003 was determined, while for the later a  $\nu$  value of 0.473 was determined.

However, different kernels namely a Gaussian (rbf), a linear and a polynomial and different regularization factors (0.1, 1, 10, 100 1000) were applied for comparison, see Fig 4.6 and Fig. 4.7. The  $\nu$ -SVR performs the best with an radial basis function kernel and a regularization factor of 1000.

Example class instantiations in python can be found below:

```
svr = SVR(kernel='rbf', degree=3, C=1000, epsilon=0.003)
```

```
nusvr = NuSVR(kernel='rbf', degree=3, C=1000, nu=0.473)
```

**Gaussian Process Regression** The last investigated method is the GPR. The method was applied with a constant regression function and an absolute exponential correlation function between two points  $\mathbf{x}$  and  $\mathbf{x}'$ , since it represents the underlying model the best. Furthermore, smoother prediction can be achieved through the usage of a nugget parameter as illustrated in Fig. 4.8.

Again, the class instantiation and assignment in python was done as stated bellow.

```
gpr = GaussianProcess(regr='constant', corr='absolute_exponential',
                      nugget=3e - 2)
```

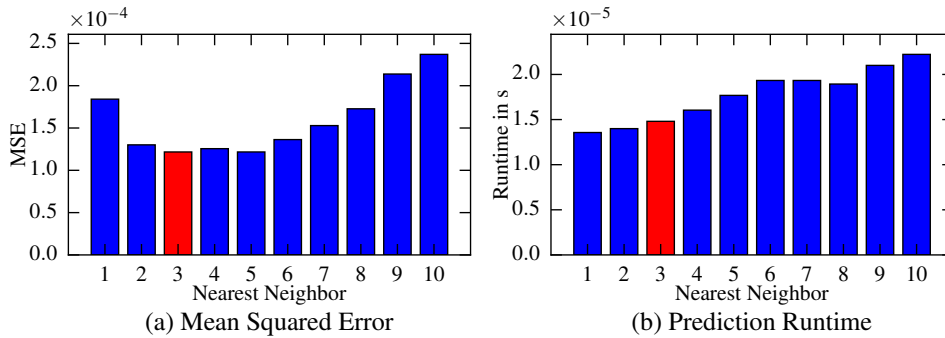


Figure 4.2: MSE and prediction runtime plots for Nearest Neighbor Regression with uniform weight function

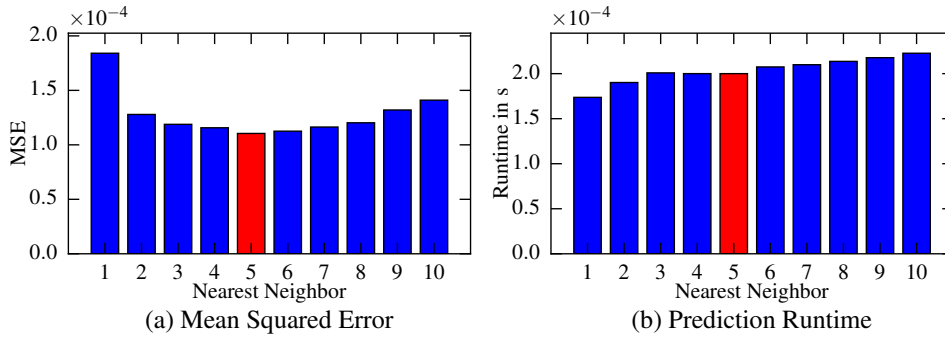


Figure 4.3: MSE and prediction runtime plots for Nearest Neighbor Regression with distance weight function

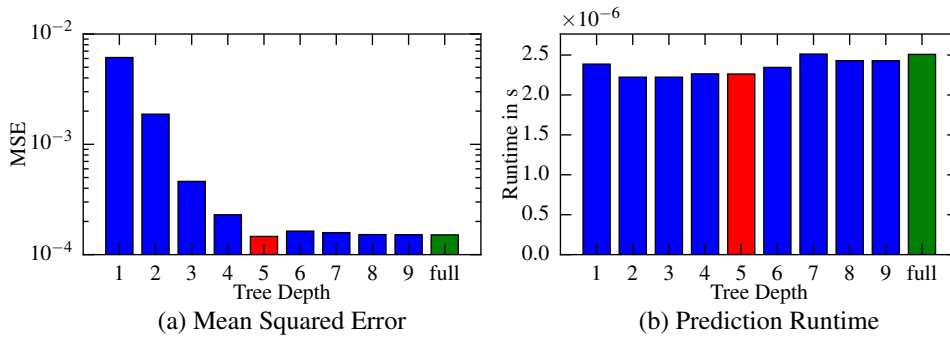


Figure 4.4: MSE and prediction runtime plots for Regression Tree

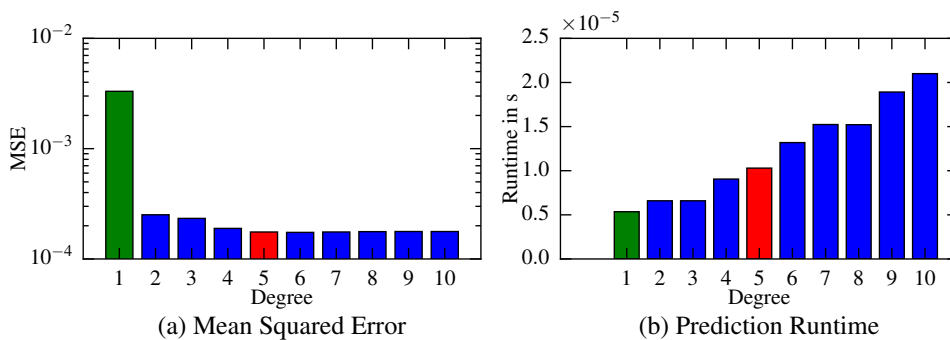


Figure 4.5: MSE and prediction runtime plots for Linear and Polynomial Regression

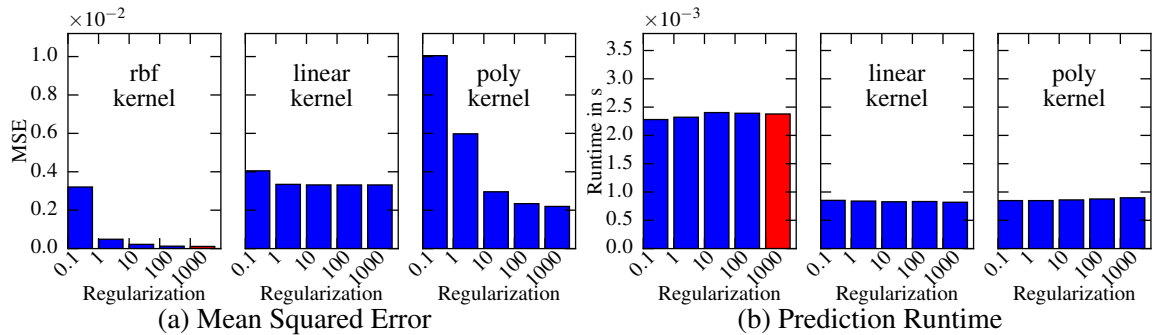
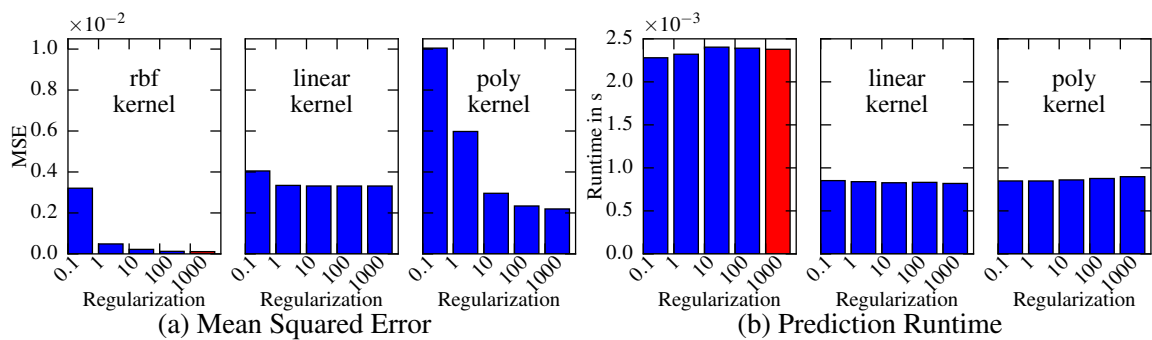
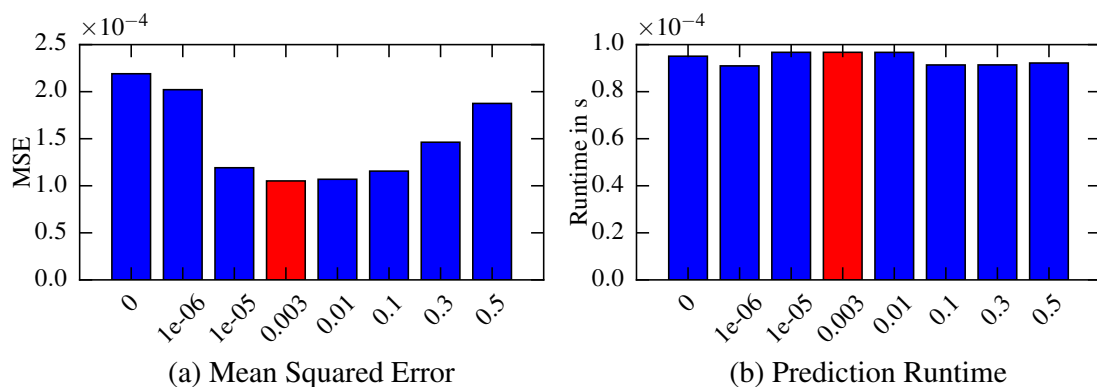
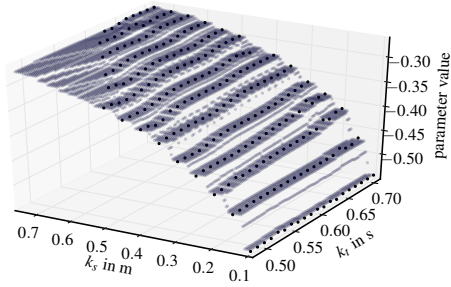
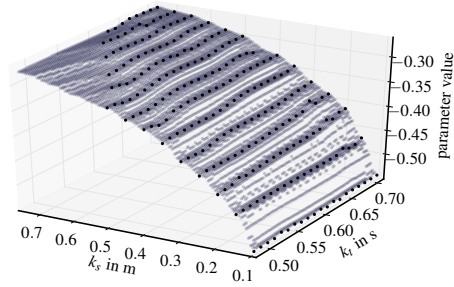
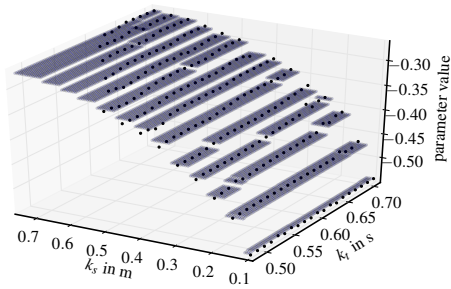
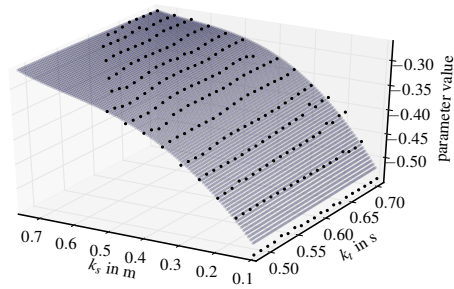
Figure 4.6: MSE and prediction runtime plots for  $\epsilon$ -Support Vector RegressionFigure 4.7: MSE and prediction runtime plots for  $\nu$ -Support Vector Regression

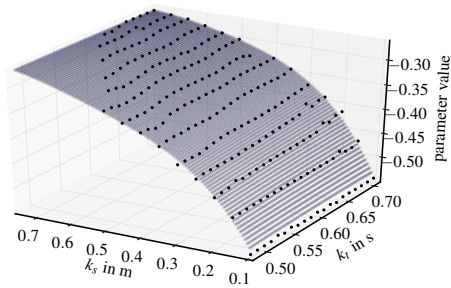
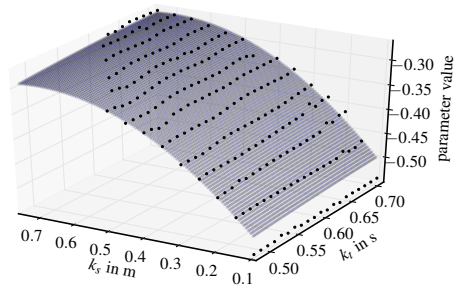
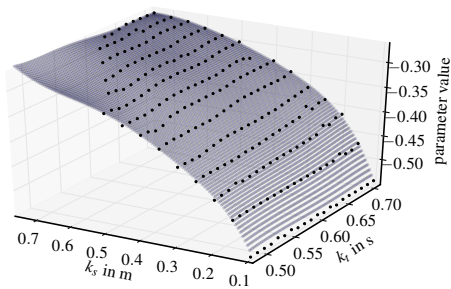
Figure 4.8: MSE and prediction runtime plots for Gaussian Process Regression

(a)  $n$ -NN with uniform weight function(b)  $n$ -NN with distance weight function

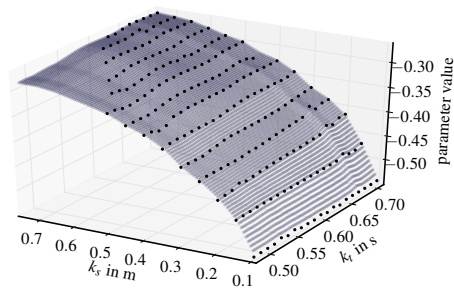
(c) Regression Tree



(d) Polynomial Regression

(e)  $\epsilon$ -Support Vector Regression(f)  $\nu$ -Support Vector Regression

(g) Gaussian Process Regression, with squared exponential correlation



(h) Gaussian Process Regression, with absolute exponential correlation

Figure 4.9: Prediction results for different machine learning methods in the range of the full task space shown for one spline parameter



## 4.2 Full Task Space Prediction

The application of machine learning methods on a data set involves the estimation of a model for the underlying function for the given data. In the previous section, the investigation of the best parametrization was performed on the *minimum velocity cost* data set. This set contains 242 samples, with 2 task parameters and 192 spline parameters. A average mean squared error and a average runtime was determined from cross-validation with 11 folds. But what does the resulting model and consequently the accuracy mean for the full task space since we only trained and tested the method on discrete task space samples? This is the topic of this section.

The resulting best parametrization for each method from chapter 4.1 (will be discussed in chapter 5 in more details), was used to fit every introduced machine learning method on the full *minimum velocity cost* data set.

The task space for the *minimum velocity cost* data set ranges from 0.1m to 0.79m in  $k_s$ , the stride-length and from 0.48s to 0.71s in  $k_t$ , the step time. The range in both direction was divided into 100 entities which resulted in 10000 samples for the prediction step to cover the full task space as intended.

In Fig. 4.9 the results for one spline parameter and every machine learning method is plotted. Here, the black dots represent the training data from the *minimum velocity cost* data set, while the blue dots are the results from the prediction of every method.

In general, the machine learning methods prediction outcome can be divided into two groups: First, there are methods that define a continuous function for the given data set, which are usually global operating methods. And second there are methods producing local plateaus, or better a discrete output. These methods operate usually locally.

A detailed analysis for every machine learning method is given below:

**Nearest Neighbor**  $n$ -NN was applied with a uniform and a distance weighting on all  $n$  nearest neighbors from a new sample in the task space. Fig. 4.9 (a) and Fig. 4.9 (b) show the prediction for the full task space respectively. The predictions between training samples from the true points for both methods results in a discrete output along the task space dimensions. The distance weighting yields a better approximation of a function with more discrete values than the uniform weighting. However, function approximations from both methods are not continuously differentiable. Another problem for  $n$ -NN regression are samples on the boundary of the task space. This method uses the  $n$  nearest neighbors on the boundary only from one side, hence the predicted value is biased to those points in contrast to a point further inside the task space, which is equally surrounded by neighbors on all sides.

**Regression Tree** The RT yields local plateaus for the outcome, which represent the average values of the samples in the leafs. Fig. 4.9 (c) illustrates the output. This method fails for the assumption of the underlying model for samples between known points from the training set.

**Polynomial Regression** The PR with a 5th order polynomial produces a reasonable underlying model assumption for the given *minimum velocity cost* data set. The predicted samples between given samples, as shown in Fig. 4.9 (d), are suitable and continuous.

**Support Vector Regression** For the SVR we have again the two case with  $\epsilon$ -SVR and  $\nu$ -SVR. Both methods approximate the underlying function equally good as shown in Fig. 4.9 (e) and Fig. 4.9 (f), respectively. A small difference in the prediction from both methods is noticeable in the corner for stride-lengths bigger than 0.55m and a step time below 0.6s, since no samples are provided from the training set.

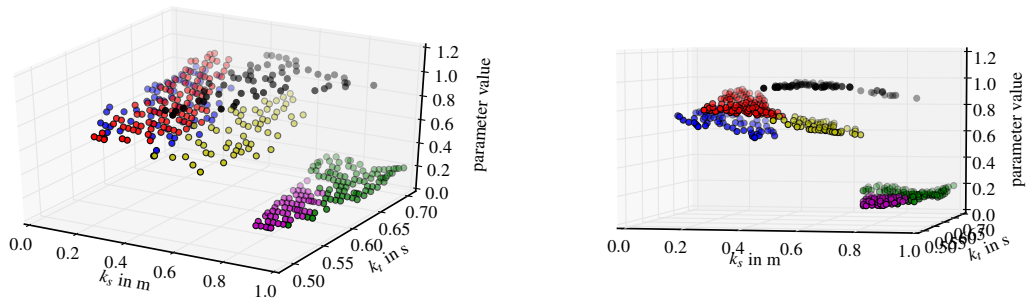
**Gaussian Process Regression** The last method is the GPR where the fitting and prediction was done with first a squared-exponential kernel, see Fig. 4.9 (g) and second an absolute-exponential kernel, see Fig. 4.9 (h). Again, since this more sophisticated method assumes a correlation between two samples, the resulting prediction for samples between the training data samples is a reasonable outcome. Although the absolute-exponential correlation function yields better results regarding the accuracy in the previous section, it seems to overfit the data. A reason for this statement is the small bump for  $k_t = 0.67s$ , which is modeled by the absolute-exponential correlation.

### 4.3 Data Clustering

The application of methods was done on two different sets. In the previous sections, the data set from the *minimum velocity cost* optimization was used where the data set was smooth for all trajectory parameters and hence the machine learning methods could be directly applied. The data set from the other cost function for minimizing the torque, however, resulted only in local smooth subsets in the task space, but an higher offset between this subsets for certain spline parameters. Therefore, a good model from the data couldn't be estimated at once. The solution to this problem was to preprocess the data set with an clustering algorithm to receive  $n$  clusters and then applying each machine learning method on every cluster subsequently.

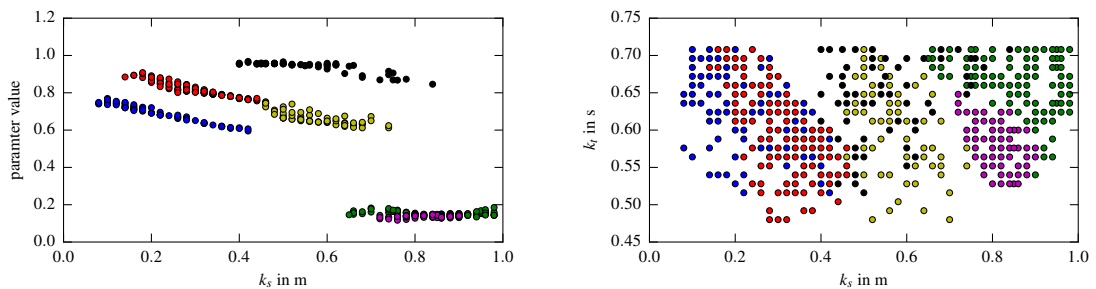
Gaussian mixture model (GMM), as introduced in chapter 3.3, was applied on the data to receive the clusters. Since, there are parameters where the subsets are more complicated to separate, GMM was applied on manually chosen parameters where a

clear distinction was possible. This resulted in better clusters than applying GMM on the full data set. An example for such a parameter is shown in Fig. 4.10 from different angles. In Fig. 4.11 (a) the offset from subsets along a task space dimension is illustrated. Initially, GMM was applied to find 12 clusters, but only cluster with more than 50 samples were chosen for the fitting step. This can be seen in 4.11 (b), where only the top 6 clusters are shown. The empty spaces are the dropped out smaller clusters, in a otherwise fully covered task space. A separation from the cost value was more difficult than in the task space on specified parameters due to a low divergence, see Fig. 4.12 and Fig. 4.13.



(a) Clusters of the torque cost data of one parameter with more than 50 samples (b) The same parameter, from a different angle

Figure 4.10: Clustering with GMM, 3D Plots



(a) 2D plot along the stride-length for a clear visualization of the clusters (b) Topview of one parameter to visualize the cluster distribution

Figure 4.11: Clustering with GMM, 2D Plots

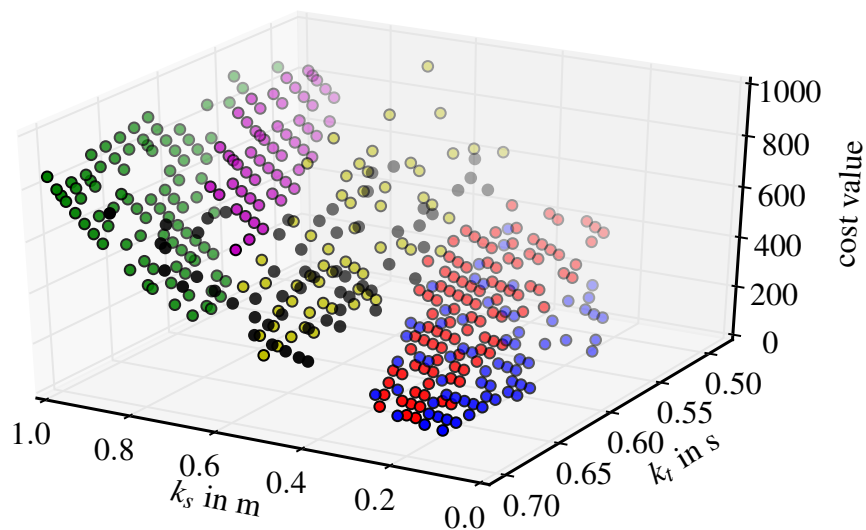


Figure 4.12: Cost value for clusters with more than 50 samples

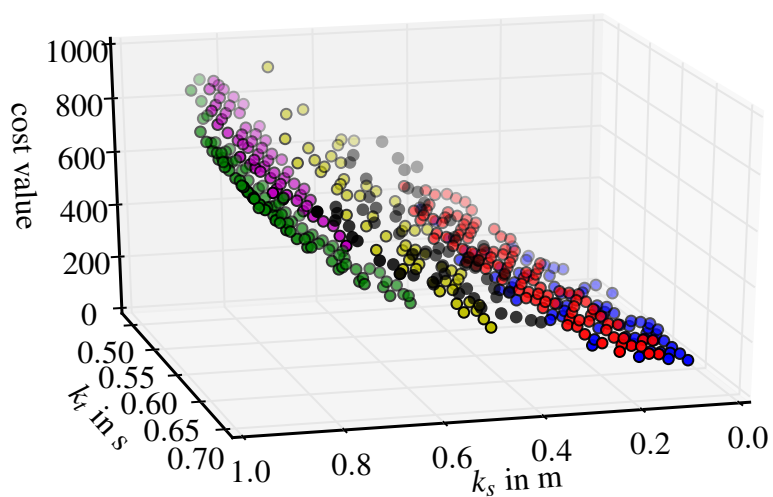


Figure 4.13: Cost value for clusters with more than 50 samples from a different angle to show difference in the cost value

# Chapter 5

## Results

After the formulation of the problem statement and the introduction of the generation of the two data sets for the velocity cost function and the torque cost function in chapter 2, the presentation of the applied machine learning methods was done in chapter 3. The results of the task in this thesis with the problem formulation to generalize optimal walking trajectories for a biped walking machine based on machine learning will be presented in this chapter.

First, the best parametrization of the machine learning methods for both data sets will be presented in chapter 5.1 since the optimal parametrization for every method depends highly on the data where the machine learning method is applied on. Second, a short presentation of the torque cost data set clustering will be presented in chapter 5.2 since this data set was not smooth in the full task space as the data set from the velocity cost function. Afterwards, the performance of the methods regarding the accuracy and the runtime is shown in chapter 5.3 and the evaluation regarding the resulting optimality loss and the constraint violation in chapter 5.4. Last but not least, the predicted trajectories were applied on a robot in a simulation environment as demonstrated in chapter 5.5.

### 5.1 Best Parametrization

Finding the best parametrization for every machine learning method for a given data set was done via a grid-search since the number of method parameters was small and often only discrete parameters present. A detailed determination was discussed in Chapter 4 for the velocity cost data set. However, the same methodology of the grid-search was proceeded on the torque cost data set. The best parametrization of the used machine learning methods like  $n$ -Nearest Neighbor ( $n$ -NN), Regression Tree (RT), Polynomial Regression (PR),  $\nu$ -Support Vector Regression ( $\nu$ -SVR) and Gaussian Process Regression (GPR) can be found in Tab. 5.1 for the velocity cost data set and in Tab. 5.2 for the torque cost data set on the next page.

Table 5.1: Best parametrization of each method for the velocity cost data set

ML-Methods	Parametrization
$n$ -NN	neighbors $n = 5$ distance weight function
RT	maxium depth $d = 5$
PR	5th order polynomial
$\nu$ -SVR	Gaussian kernel $\nu = 4.73 \cdot 10^{-1}$ $C = 1 \cdot 10^3$
GPR	absolute exponential correlation constant regression function nugget = $3 \cdot 10^{-2}$

Table 5.2: Best parametrization of each method for the torque cost data set

ML-Methods	Parametrization
$n$ -NN	neighbors $n = 3$ distance weight function
RT	maxium depth $d = 5$
PR	4th order polynomial
$\nu$ -SVR	Gaussian kernel $\nu = 3.0 \cdot 10^{-1}$ $C = 1 \cdot 10^3$
GPR	absolute exponential correlation constant regression function nugget = 0

## 5.2 Clustering of the Torque Cost Data Set

Due to reasons described in chapter 4.3, the torque cost data set was preprocessed with a clustering algorithm to determine the local smoother subsets for the spline parameter values for the full task space. Although GMM with 12 components was applied, a dexterous merge of nearby smaller clusters yielded in the end 4 big cluster with 197, 163, 108 and 59 samples as shown in Fig. 5.1 (a) and (b), whereas distinct clusters can be seen for the parameter in (a) it is impossible to determine the cluster for the parameter in (b). A clear distinction along a task space dimension (stride-length  $k_s$ ) of the 4 cluster for one spline parameter is shown in Fig. 5.2 (a) and a not clear distinction for the parameter in (b).

Since the clusters are overlapping in the task space, the prediction of the spline parameter  $\mathbf{p}$  for new task space parameter  $\mathbf{k}$  was done on an estimated model from a cluster with a lower cost value.

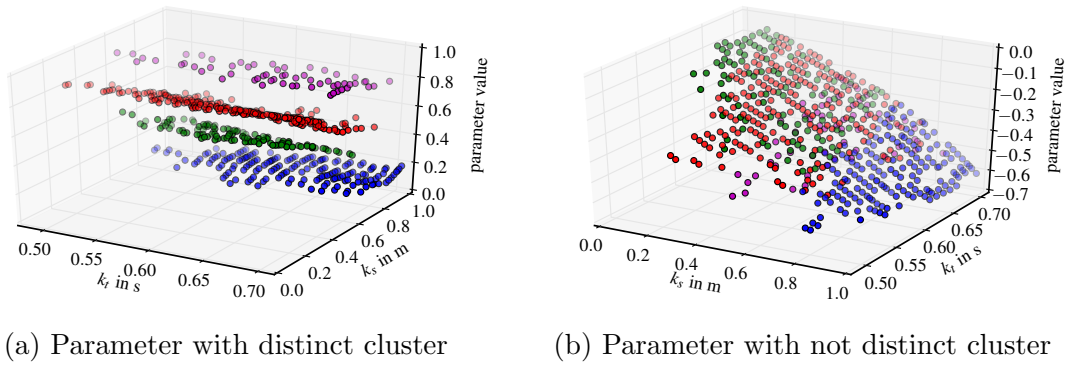


Figure 5.1: Clustering of two parameter from the torque cost data set with resulting 4 biggest cluster

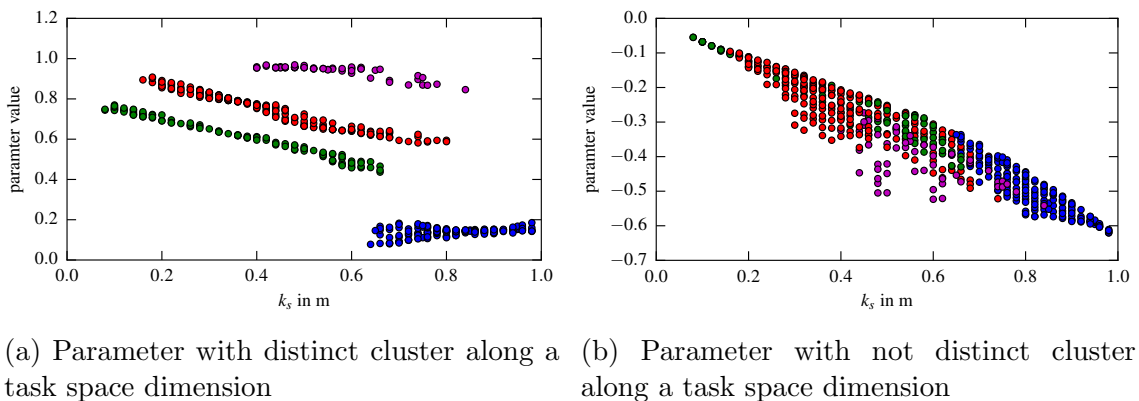
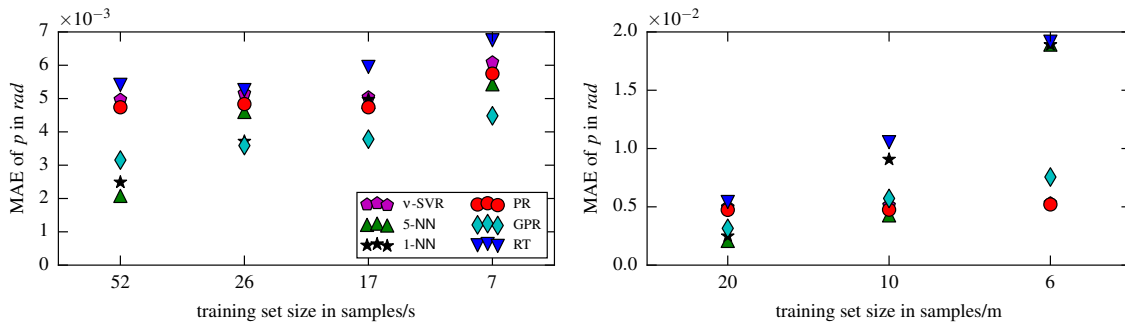


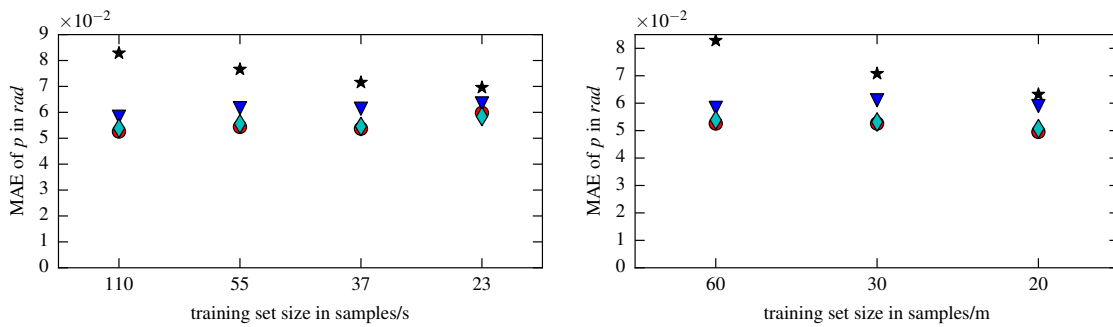
Figure 5.2: 2D Plot of the 4 biggest cluster for two parameter from the torque cost data set



(a) Different sampling densities in the step time direction

(b) Different sampling densities in the stride-length direction

Figure 5.3: MAE for predicted spline parameters from the velocity cost data set



(a) Different sampling densities in the step time direction

(b) Different sampling densities in the stride-length direction

Figure 5.4: MAE for predicted spline parameters from the torque cost data set

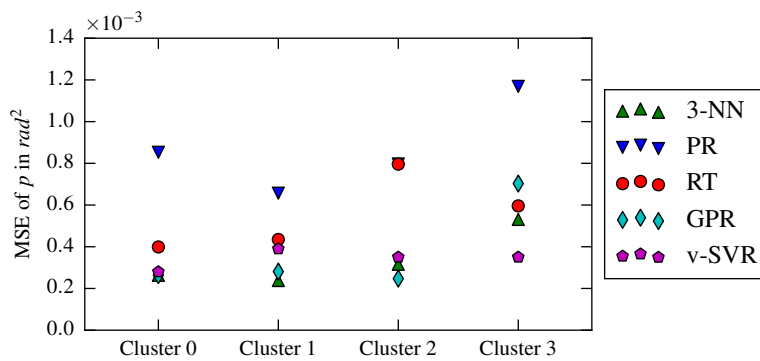


Figure 5.5: MSE per cluster in the torque cost data set



## 5.3 Machine Learning Methods Performance

The underlying model for the 192 trajectory parameter was estimated with different machine learning methods on the two data sets to evaluate the quality of the mapping  $\mathbf{k} \rightarrow \mathbf{p}$  with 11 fold cross-validation. The accuracy and the runtime of the 5 machine learning methods RT, PR, GPR,  $n$ -NN and the  $\nu$ -SVR were compared against each other. Furthermore, a look-up table in the form of a 1-NN was also applied to compare this straight forward approach to the more advanced machine learning methods. The mean absolute error for both data sets can be seen in Fig. 5.3 and Fig. 5.4. The first entries for the plots in Fig. 5.3 (a) and (b) and Fig. 5.4 (a) and (b) are results from the full set, while the following entries are reduced subsets along the corresponding task space dimension. The runtime was estimate from an average of 1000 predictions of a full trajectory parameter set on a Intel Core i7 CPU.

### 5.3.1 Velocity Cost Data Set

The velocity cost data set consists of 242 samples per parameter, resulting in a density of 52 samples per second in the task space dimension for the step time and with a density of 20 samples per meter in the task space dimension for the stride-length.

**Accuracy** In general all methods have had an acceptable MAE for our task, while the reduction of the sample density along the step time had a smaller influence on the error then the subsampling of the stride-length.

**Runtime** The runtime of every method (in seconds) regarding the number of samples is shown in Tab. 5.3. All methods are ordered with the fastest method at the top.

### 5.3.2 Torque Cost Data Set

The torque cost data set consists of 661 samples per parameter in the full set while after clustering the set was reduced to 523 samples in the 4 clusters combined. This data set was denser due to the more complicated cost function and hence known fact of the non-smooth surfaces for certain parameter. In this set there was a maximum of 110 samples per second in the task space dimension for the step time and a maximum of 60 samples per meter in the task space dimension for the stride-length.

**Accuracy** The MAE for the torque cost data set was higher than for the previous set since the methods were directly applied on the full set without clustering. The higher error was due to the known issue that some parameters were not smooth in the full task space.

**Cluster** Since the torque cost data set was also divided into four cluster, a model accuracy for every method per cluster was investigated. The resulting mean squared error in Fig. 5.5 was as expected lower through the application of every method per cluster than the application on the full set.

**Runtime** The prediction runtime (in seconds) per cluster is shown in Tab. 5.4 with the same ordering of the methods as before.

Table 5.3: Runtime for the prediction of a trajectory from the velocity cost data set

ML-Methods	Samples in training set			
	220	110	55	28
RT	$2.3 \cdot 10^{-6}$	$2.3 \cdot 10^{-6}$	$2.3 \cdot 10^{-6}$	$2.5 \cdot 10^{-6}$
PR	$1.1 \cdot 10^{-5}$	$1.1 \cdot 10^{-5}$	$1.1 \cdot 10^{-5}$	$1.0 \cdot 10^{-5}$
GPR	$6.2 \cdot 10^{-5}$	$4.1 \cdot 10^{-5}$	$3.4 \cdot 10^{-5}$	$2.6 \cdot 10^{-5}$
$n$ -NN	$2.1 \cdot 10^{-4}$	$2.2 \cdot 10^{-4}$	$2.1 \cdot 10^{-4}$	$2.2 \cdot 10^{-4}$
$\nu$ -SVR	$2.5 \cdot 10^{-3}$	$1.6 \cdot 10^{-3}$	$1.3 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$

Table 5.4: Runtime for the prediction of a trajectory from the torque cost data set

ML-Methods	Cluster			
	0	1	2	3
RT	$3.0 \cdot 10^{-6}$	$5.0 \cdot 10^{-6}$	$6.0 \cdot 10^{-6}$	$1.2 \cdot 10^{-5}$
PR	$1.4 \cdot 10^{-5}$	$1.9 \cdot 10^{-5}$	$1.4 \cdot 10^{-5}$	$2.8 \cdot 10^{-5}$
GPR	$8.2 \cdot 10^{-5}$	$7.5 \cdot 10^{-5}$	$7.5 \cdot 10^{-5}$	$8.7 \cdot 10^{-5}$
$n$ -NN	$2.1 \cdot 10^{-4}$	$2.5 \cdot 10^{-4}$	$3.5 \cdot 10^{-4}$	$6.4 \cdot 10^{-4}$
$\nu$ -SVR	$1.5 \cdot 10^{-3}$	$1.2 \cdot 10^{-3}$	$1.9 \cdot 10^{-3}$	$2.2 \cdot 10^{-3}$

## 5.4 Evaluation of the Predicted Trajectory Parameters

The evaluation of the machine learning methods presented a relative low model error for both data sets on almost all methods. Furthermore, the trajectory parameter prediction runtime was at least (often even more) by a factor of 10 below the targeted limit of 30ms, which was chosen to be applicable in real-time tasks. But what does this mean for the task of biped walking? This is going to be investigated in this chapter.

First, the relative optimality loss regarding the cost value determined by one of the two cost functions was computed for the predicted trajectories. The optimality loss was computed relative to the cost value for the trajectory from the optimization. Second, due to an error from the model estimation, the resulting trajectories are analysed wrt the constraint violation. The interval width of the inequality constraints was used to normalize the constraint violation to be able to compare different constraints.

The relative optimality loss and the normalized constraint violation for the velocity cost data set is shown in Fig. 5.6 and for the torque cost data set in Fig. 5.7 in the top row and the bottom row of the figure respectively.

Again, the first entry in every plot corresponds to the full set, while the following entries are subsets for one of the two task space dimensions.

### 5.4.1 Velocity Cost Data Set

**Cost Value** The resulting relative optimality loss for the velocity cost data set is for some predicted trajectory parameter  $\mathbf{p}_{predicted}$  negative. One reason for this is that the given task space parameter  $\mathbf{k}$  are not achieved and hence a smaller or slower step is executed.

**Constraint Violation** Since a conservative bounding of the constraint was chosen in the optimization the resulting normalized constraint violations from the prediction for all methods are within a reasonable range.

### 5.4.2 Torque Cost Data Set

**Cost Value** As with the velocity cost data set, the relative optimality loss for the predicted trajectory parameter from the torque cost data set resulted mostly in negative values due to the reason as mentioned before.

**Constraint Violation** As before, the conservative bounding of the constraints in the optimization resulted in small normalized constraint violations.

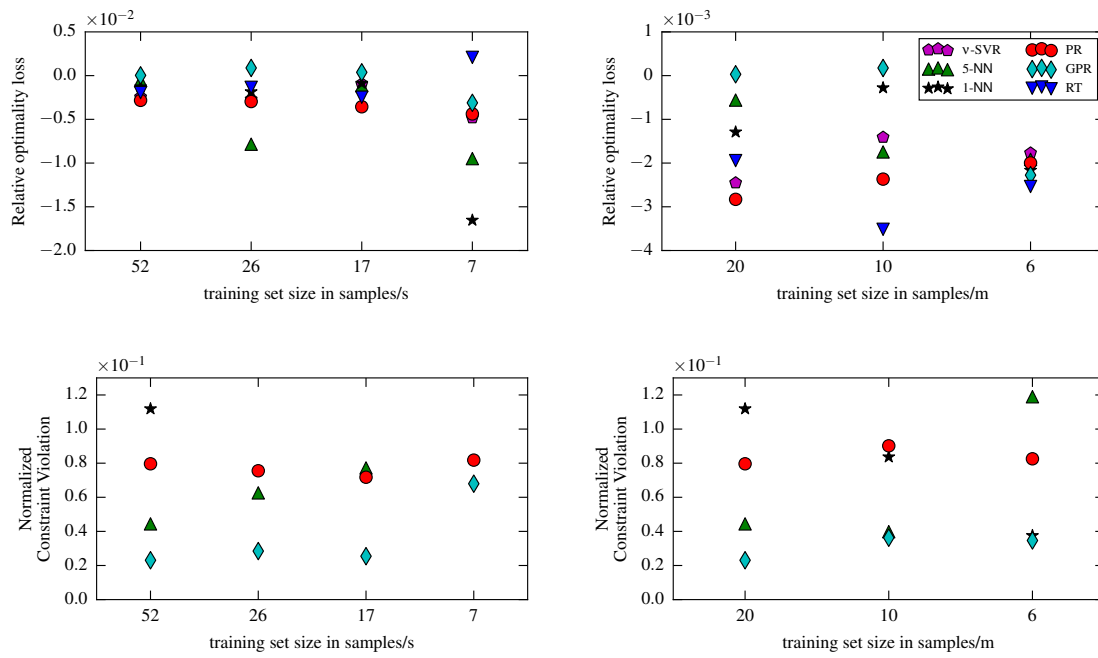


Figure 5.6: Evaluation of predictions from the velocity cost data set

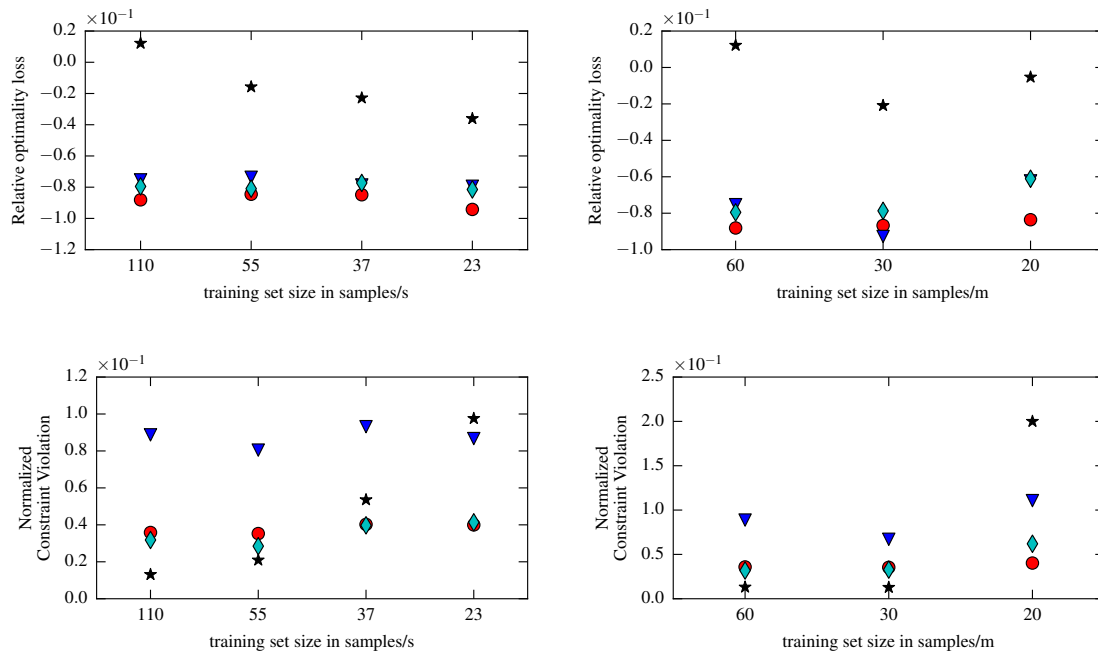


Figure 5.7: Evaluation of predictions from the torque cost data set

## 5.5 Simulation with Predicted Parameter

The final step in the task of generalizing walking trajectories from the data sets was the application of the predicted trajectory parameter in a simulation environment to show the feasibility of the approach. A popular rigid-body simulation environment among researcher in the field of robotics is OpenHRP [KHK04] which was used to evaluate the predicted trajectory parameter. Furthermore, OpenHRP can interface with MATLAB/Simulink to allow the application of a walking controller.

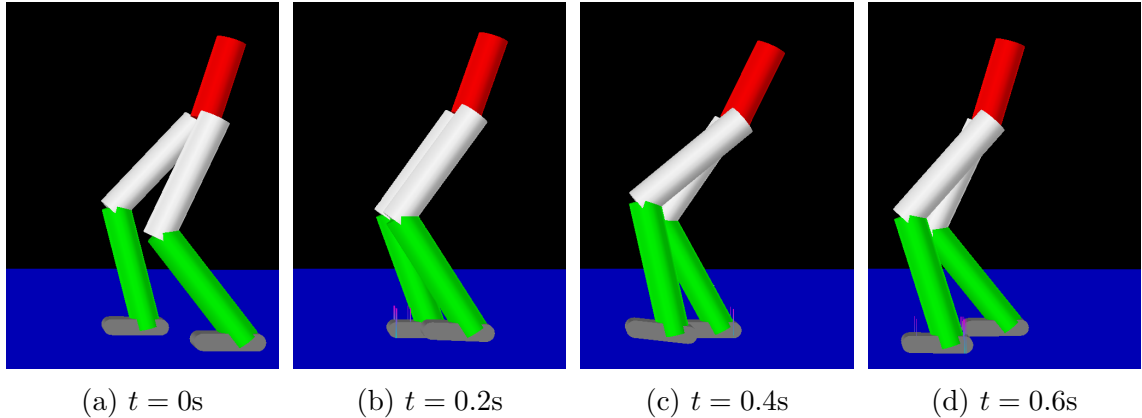


Figure 5.8: Walking simulation of a biped robot in OpenHRP

A biped walker model and a custom walking controller developed at the DLR were used to show the execution of a step for a given task space parameter  $\mathbf{k}$  as illustrated in Fig. 5.8. The four snapshots were taken 0.2s apart for a full step. In this example, the right foot was the stance foot and the left one the swing foot.

The simulation shows that a feasible walking trajectory from predicted trajectory parameter can be executed.

**Transition between different steps** To allow the application of non-cyclic gaits, a transition between motion primitives was developed.

$$q_{transition} = \sigma(\beta)q_a(\mathbf{p}_a, t) + [1 - \sigma(\beta)]q_b(\mathbf{p}_b, t) \quad (5.1)$$

The transition function in Eq. 5.1 allows the transformation of the trajectory  $q_a$  with the parameter  $\mathbf{p}_a$  from one step to the walking trajectory  $q_b$  with the parameter  $\mathbf{p}_b$  of another step. This can be done during one or several steps and is determined by the sigmoid function  $\sigma(\beta)$ .



## Chapter 6

# Conclusion & Outlook

The task in this thesis, to generalize optimal walking trajectories from non-linear optimization for a biped walking machine was achieved. This was done on two different data sets from a velocity cost function  $\Gamma_{\dot{q}}$  and a torque cost function  $\Gamma_{\tau}$ . The velocity cost data set was smooth for all trajectory parameter in the full task space. The torque cost data set was however more complex structured and some parameter hadn't continous regions. To solve the problem with the torque cost data set, two additional steps were done: First, the generation of a more denser data set and second, a preprocessing with a clustering algorithm. The resulting four clusters were fitted individually and for cluster with overlapping regions in the task space, the one with the lower cost value chosen for new predictions.

All machine learning methods for regression problems were accordingly parameterized to fit the data as good as possible while maintaining generalization. The performance for all methods was good on the velocity cost data set and comparably well on the cluster of the torque cost data set. The Gaussian Proces Regression method had the best overall performance regarding the error and runtime and hence the resulting constraint violations. The targeted limit for the prediction time of 30ms was achieved. This allows the prediction of several walking trajectories for different task space parameters for the next steps.

Future work can target the generalization of walking trajectories in 3D and a biped walker with more degrees of freedom. Additionally, since the fast prediction of the machine learning allows the computation of several trajectories, a task planner on a higher level can take use of this to provide robust walking for unforeseen disturbances.





# List of Figures

1.1	TORO, the torque-controlled humanoid robot (DLR) . . . . .	2
1.2	Generalization of optimal walking trajectories pipeline . . . . .	5
2.1	Non-linearity of a paramter from the torque cost data set . . . . .	9
3.1	Machine Learning Approaches and Categories . . . . .	12
3.2	Example Data Set . . . . .	14
3.3	$n$ -NN Regression Example . . . . .	16
3.4	Regression Tree . . . . .	17
3.5	Applied Regression Tree Example . . . . .	18
3.6	Linear/Polynomial Regression Example . . . . .	20
3.7	Epsilon tube and slack variable . . . . .	21
3.8	Support Vector Regression Example . . . . .	24
3.9	Gaussian Process Regression Example . . . . .	27
4.1	Grid Search . . . . .	32
4.2	MSE & Runtime for $n$ -NN (uniform weight) . . . . .	34
4.3	MSE & Runtime for $n$ -NN (distance weight) . . . . .	34
4.4	MSE & Runtime for RT . . . . .	34
4.5	MSE & Runtime for PR . . . . .	34
4.6	MSE & Runtime for $\epsilon$ -SVR . . . . .	35
4.7	MSE & Runtime for $\nu$ -SVR . . . . .	35
4.8	MSE & Runtime for GPR . . . . .	35
4.9	Prediction for Full Task Space . . . . .	36
4.10	Clustering with GMM, 3D Plots . . . . .	39
4.11	Clustering with GMM, 2D Plots . . . . .	39
4.12	Cost Value for Clusters . . . . .	40
4.13	Cost Value for Clusters . . . . .	40
5.1	Clustering with GMM, 3D Plot with 4 Cluster . . . . .	43
5.2	Clustering with GMM, 2D Plot with 4 Cluster . . . . .	43
5.3	MAE for predicted spline parameters from the velocity cost data set . . . . .	44
5.4	MAE for predicted spline parameters from the torque cost data set . . . . .	44
5.5	MSE per cluster in the torque cost data set . . . . .	44

5.6	Evaluation of predictions from the velocity cost data set . . . . .	48
5.7	Evaluation of predictions from the torque cost data set . . . . .	48
5.8	Walking simulation of a biped robot in OpenHRP . . . . .	49

# Bibliography

- [Alp14] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2014.
- [B<sup>+</sup>06] Christopher M Bishop et al. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.
- [BWS<sup>+</sup>09] Christoph Borst, Thomas Wimböck, Florian Schmidt, Matthias Fuchs, Bernhard Brunner, Franziska Zacharias, Paolo Robuffo Giordano, Rainer Konietzschke, Wolfgang Sepp, Stefan Fuchs, et al. Rollin’justin-mobile platform with variable base. In *ICRA*, pages 1597–1598, 2009.
- [CH67] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.
- [CL02] Chih-Chung Chang and Chih-Jen Lin. Training nu-support vector regression: theory and algorithms. *Neural Computation*, 14(8):1959–1978, 2002.
- [DBK<sup>+</sup>97] Harris Drucker, Chris JC Burges, Linda Kaufman, Alex Smola, Vladimir Vapnik, et al. Support vector regression machines. *Advances in neural information processing systems*, 9:155–161, 1997.
- [Ebd08] Mark Ebden. Gaussian processes for regression: A quick introduction. *The Website of Robotics Research Group in Department on Engineering Science, University of Oxford*, 2008.
- [EWO<sup>+</sup>] Johannes Engelsberger, Alexander Werner, Christian Ott, Bernd Henze, Maximo A Roa, Gianluca Garofalo, Robert Burger, Alexander Beyer, Oliver Eiberger, Korbinian Schmid, et al. Overview of the torque-controlled humanoid robot toro.
- [HKK<sup>+</sup>04] Hirohisa Hirukawa, Fumio Kanehiro, Kenji Kaneko, Shuuji Kajita, Kiyoshi Fujiwara, Yoshihiro Kawai, Fumiaki Tomita, Shigeoki Hirai, Kazuo Tanie, Takakatsu Isozumi, et al. Humanoid robotics platforms developed in hrp. *Robotics and Autonomous Systems*, 48(4):165–175, 2004.

- [HTFF05] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. *The elements of statistical learning: data mining, inference and prediction*, volume 27. Springer, 2005.
- [JWHT13] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*. Springer, 2013.
- [KHK04] Fumio Kanehiro, Hirohisa Hirukawa, and Shuuji Kajita. Openhrp: Open architecture humanoid robotics platform. *The International Journal of Robotics Research*, 23(2):155–165, 2004.
- [LBU09] Sebastian Lohmeier, Thomas Buschmann, and Heinz Ulbrich. System design and control of anthropomorphic walking robot lola. *Mechatronics, IEEE/ASME Transactions on*, 14(6):658–666, 2009.
- [LNTC<sup>+</sup>11] Roberto Lampariello, Duy Nguyen-Tuong, Claudio Castellini, Gerd Hirzinger, and Jan Peters. Trajectory planning for optimal robot catching in real-time. In *International Conference on Robotics and Automation (ICRA)*, pages 3719–3726, 2011.
- [Mit97] Tom M Mitchell. *Machine learning*. WCB. McGraw-Hill Boston, MA:, 1997.
- [Mur12] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.
- [OAS<sup>+</sup>06] Yu Ogura, Hiroyuki Aikawa, Kazushi Shimomura, Akitoshi Morishima, Hun-ok Lim, and Atsuo Takanishi. Development of a new humanoid robot wabian-2. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 76–81. IEEE, 2006.
- [PVG<sup>+</sup>11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [Rey09] Douglas Reynolds. Gaussian mixture models. In *Encyclopedia of Biometrics*, pages 659–663. Springer, 2009.
- [RW06] Carl Rasmussen and Chris Williams. Gaussian processes for machine learning. *Gaussian Processes for Machine Learning*, 2006.
- [SS04] Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

- [SSWB00] Bernhard Schölkopf, Alex J Smola, Robert C Williamson, and Peter L Bartlett. New support vector algorithms. *Neural computation*, 12(5):1207–1245, 2000.
- [VB04] Miomir Vukobratovic and Branislav Borovac. Zero-moment point-thirty five years of its life. *International Journal of Humanoid Robotics*, 1(01):157–173, 2004.
- [VGS97] Vladimir Vapnik, Steven E Golowich, and Alex Smola. Support vector method for function approximation, regression estimation, and signal processing. *Advances in neural information processing systems*, pages 281–287, 1997.
- [WHHAS13] Roman Weitschat, Sami Haddadin, Felix Huber, and A Albu-Schauffer. Dynamic optimality in real-time: A learning framework for near-optimal robot motions. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 5636–5643, 2013.
- [WLO12] Alexander Werner, Roberto Lampariello, and Christian Ott. Optimization-based generation and experimental validation of optimal walking trajectories for biped robots. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 4373–4379, 2012.
- [WLO14] Alexander Werner, Roberto Lampariello, and Christian Ott. Trajectory optimization for walking robots with series elastic actuators. In *53rd Conference on Decision and Control (CDC)*, pages 2964–2970, 2014.