Gaussian Process Regression for Data-Driven Model Predictive Control

Nuno António Seco Rodrigues

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Gaussian Process Regression for Data-Driven Model Predictive Control

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Nuno António Seco Rodrigues

June 6, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Gaussian Process Regression for Data-Driven Model Predictive Control

by

Nuno António Seco Rodrigues

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: June 6, 2021

Supervisor(s): _____
prof.dr.ir. Jan-Willem van Wingerden

Reader(s): _____

Abstract

Systems and Control deals with modelling and control design of many different types of systems with different behaviour and characteristics. Often times it is challenging to build mathematical models to describe the system dynamics due to limited knowledge of the underlying system, complex nonlinearities, or the existence of unknown exogenous phenomena impacting the input/output response. For this reason, it's important to develop methods of System Identification to learn these dynamics using only batches of input/output data, as well as control strategies capable of handling such models.

The necessity for flexibility in the model and ability to deal with nonlinear mappings has led to the interest of applying machine learning methods to Identification and control design. This thesis focuses on using Gaussian Processes as a means to learn a model from batch training data and on developing a control strategy capable of handling such a model. The goal is to propose a Data-driven algorithm for closed-loop control and to explore the properties of a Gaussian Process that make it a candidate method for control applications.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank professor Jan-Willem van Wingerden for supervising this thesis. For his guidance, pragmatism, and for giving me the opportunity to begin this research work several months ago during a time when I was struggling to get started.

I also want to thank Frederiek Wesel for lending me his support and expertise throughout the project and for always being available to answer questions and give me advice.

Whatever I accomplished with this thesis has been improved by the contributions of these two people.

Delft, University of Technology
June 6, 2021

Nuno António Seco Rodrigues

Chapter 1

Introduction

The process of controlling a dynamic system can be carried out in several ways. Different controlling techniques can be lumped into Model-based control approaches and Data-driven control approaches. A Model-based control design makes use of previous knowledge of the process model and determination of a controller transfer function, but it often requires deriving mathematical models from a first principles approach. This can be a challenge if there is little information regarding the system, if the system exhibits nonlinear behaviour, or if there are unknown disturbance effects affecting the input/output response.

Data-driven control design methods make use of batches of input/output data taken from the process plant. The information on this observed data is used to fit the data to a model structure and estimate values for the parameters of a controller with a fixed transfer function [1]. The problem with Data-driven methods is that to extrapolate information on the input/output behaviour of a system it is always required to make assumptions about the underlying process. These assumptions, such as linearity of the system or a fixed parametrization of the transfer function, can often be too narrowing and result in poor control performance if they do not agree with the characteristics of the unknown process plant.

In order to increase flexibility of the model while using input/output data, one alternative of the current state-of-the-art algorithms is to learn the system dynamics by using Machine Learning methods such as Neural Networks [2], Random Decision Forests [3], Support Vector Machines [4] or Gaussian Processes [5]. These methods are less restrictive in their prior assumption regarding the learning data, and under certain circumstances can be shown to be universal approximators [6]. It's possible to use such methods to infer prediction models using only input/output data, similarly to the aforementioned Data-driven approaches, while taking benefit of the flexibility that Machine Learning algorithms provide.

These Machine Learning methods used for System Identification are often combined with Model Predictive Control (MPC), which is an optimal Model-based control technique capable of handling general nonlinear models. Performing System Identification using Machine

Learning methods trained from input/output data in combination with MPC has been liberately named Data-driven Model Predictive Control (Data-driven MPC) (e.g. in [7]). While there are technical differences[1], the model is obtained solely from experimental data and the controller tuning parameters using MPC can be carried out intuitively based on performance specifications, which makes Data-driven MPC algorithms extremely flexible practical tools.

1-1 Research Overview

In the recent few years Neural Networks have been the target of much academic research (e.g. [8],[9], [10]). Gaussian Processes have seen considerably less studied (e.g. *CiteSeerX*, one of the largest research repositories in the world, shows five times more results for Neural Networks). However, Gaussian Process Regression has been shown to outperform Neural Networks on several benchmark simulations (see [11]), and Gaussian Processes hold unique properties that make them stand out.

Gaussian Processes as prediction models in the MPC framework were proposed in [12]. However, more recent research (see e.g. [13], [14], [15]) mainly uses them as an augmentation for a physics-based model in order to estimate unexpected disturbances. Furthermore, many approximate the stochastic prediction to deterministic values in order to reduce computational costs. While this makes sense, crucial information of the model is lost in the simplification process.

Gaussian Process models have also been used in conjunction with Feedback Linearization approaches (e.g. [16], [17]). While these algorithms try to make use of the full predictive capabilities of the Gaussian Process model, they do not use MPC and as such the solution is not mathematically optimal under a given cost function, and properties of the MPC such as ability to handle nonlinear state and input constraints cannot be implemented.

1-2 Research questions

Section 1-1 sects the stage for the research focus of this thesis. The main contributions of this thesis are the use of Gaussian Processes for System Identification as the prediction model without making use of any physical-based model or prior knowledge on the system; and the incorporation of the model's unique properties and extra information in the MPC framework. The main questions that this research attempts to answer are:

1. Can Gaussian Processes be used as a model for System Identification using only batch input/output data? Does the model achieve reasonable prediction results?

[1] It differs from traditional Data-driven approaches in the sense that Identification and controller design are two distinct steps. Conversely, in Data-driven algorithms such as [1] the input/output data leads directly to controller design (the Identification process is embedded in the algorithm).

2. Can the model be used in a (Nonlinear) MPC framework?

3. Compare the above mentioned control performance to MPC using more standard deterministic models, in terms of advantages and disadvantages. Particularly, can one take advantage of useful properties unique to the Gaussian Process, such as the stochastic Gaussian predictions and the way this method makes use of the training data as a *linear smoother*?

4. What are the potential barriers in implementation, such as the performance dependence on the training data, the required prior knowledge and the ability to scale the problem to more complex systems and larger data sets?

5. Can the final algorithm be applied in practice to both simulations and physical set-ups? What are the challenges in practical implementation?

1-3 Thesis outline

The structure of this thesis is the following: Chapter 2 introduces the basics of Gaussian Process Regression; Chapter 3 discusses how to use Gaussian Processes to the specific problem of System Identification, including the Identification of a model, using the model to make predictions and adaptation to online applications; Chapter 4 discusses Gaussian Process approximations as a way of dealing with large data sets, which is the main limitation of the method; Chapter 5 introduces MPC as the preferred controller method and discusses adaptations to the standard formulation that allow it to incorporate the Gaussian Process model. In the end, a Data-driven MPC algorithm is introduced; Chapter 6 implements the aforementioned algorithm to a variety of test cases, using both system simulations and an application on a physical system setup; Chapter 7 concludes the thesis, attempts to answer the questions from Section 1-2 and presents proposals for future research work.

Chapter 2

Overview of Gaussian Process Regression

This Chapter briefly introduces the theory behind a Gaussian Process (GP) and its applications in the problem of Regression. This is not meant to be an extensive introduction on the topic, but rather an overview of the methods which segues into the topics of the next Chapters. A more extensive overview on the topic can be found in [18].

Given a finite data set of N points $\{\boldsymbol{x}_i, y_i\}, \forall i \in \{1, ..., N\}$, where $\boldsymbol{x}_i$ a function input[1] and y_i the respective function output, the problem of Regression deals with learning the mapping function $f(\cdot)$ from the inputs to the outputs. There are several ways to perform Regression, and they all make assumptions on certain properties of the function $f(\cdot)$. One form of Regression is to assume that the function is linear in the inputs, and a method such as least-squares can be used (see [19]). This method is very popular in applications because it is simple and effective, and it often gives an explicit solution that is easy to compute. However, many functions cannot be approximated in a linear way without resulting in high bias, because they depend on their inputs through nonlinear mappings.

All Regression methods have to make assumptions on some of the underlying properties of the unknown function. Gaussian Process Regression (GPR) is a method for nonlinear Regression which assumes from the beginning that the function can be modelled as a Gaussian Process, which is a concept introduced next.

2-1 Inference with Gaussian Processes

Consider the values y_i as observations of the function $f(\boldsymbol{x})$ at input locations $\boldsymbol{x}_i$ corrupted by i.i.d. Gaussian noise ϵ_i, that is

[1] Remark on notation: This thesis uses bold letters (e.g. $\boldsymbol{a}$) for vectors and capital letters (e.g. A) for matrices

$$y_i = f(\boldsymbol{x}_i) + \epsilon_i, \qquad \epsilon_i \sim \mathcal{N}(0, \sigma_n^2) \ . \tag{2-1}$$

The inputs $\boldsymbol{x}_i$ are taken as vectors $\boldsymbol{x}_i \in \mathbb{R}^{D\times 1}$ and the outputs as scalars $y_i \in \mathbb{R}$. The prior assumption put on the function $f(\cdot)$ is that every finite number of function values f_i (f_i here is short-hand notation for $f(\boldsymbol{x}_i)$) has a multi-variable joint Gaussian distribution. A GP is by definition a collection of random variables, any subset of which has a Gaussian distribution. We make the prior assumption (prior in the sense that it is an assumption made before considering the data) that the function $f(\boldsymbol{x})$ is a GP.

A GP is fully defined by a mean function $m(\boldsymbol{x})$ and a covariance function $k(\boldsymbol{x}, \boldsymbol{x'})$:

$$f(\boldsymbol{x}) = \mathcal{N}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x'})) \ . \tag{2-2}$$

All the available data can be represented in a data set $D = \{X, \boldsymbol{y}\}$, where $X = [\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N]^T \in \mathbb{R}^{N\times D}$, $\boldsymbol{y} = [y_1, y_2, ..., y_N]^T \in \mathbb{R}^{N\times 1}$ and N is the number of data points available. In order to abide by the standard terminology of the literature regarding Machine Learning Regression problems, inputs $\boldsymbol{x}_i$ to the GP will be called regressors and outputs y_i will be called targets.

To simplify notation, the vector of training values $f(X) \in \mathbb{R}^{N\times 1}$ corresponding to the true values of the function evaluated at the training inputs is written as $\boldsymbol{f}$ and the true values of unknown (possibly multiple) query points, or prediction points, $f(X_*)$ is written as $\boldsymbol{f_*}$. These values of the unknown function have a probabilistic joint Gaussian distribution given by the prior assumption

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f_*} \end{bmatrix} = N\left(\begin{bmatrix} \boldsymbol{m_f} \\ \boldsymbol{m_*} \end{bmatrix}, \begin{bmatrix} K_{ff} & K_{f*} \\ K_{*f} & K_{**} \end{bmatrix} \right) \ . \tag{2-3}$$

Defining the prior mean and covariance functions means encoding prior beliefs about the function. The mean function $m(\cdot) : \mathbb{R}^D \mapsto \mathbb{R}$ is considered to be $m(\boldsymbol{x}) = 0$ from this point forward. This is done for simplicity, although it is possible to consider different prior function means (see [18]). In general, taking the prior mean to be zero is common for most applications, since the target data can always be normalized to have a zero mean. This predictive mean value is not of great importance since we will later condition the function to the target values used for training. In practice, assuming a zero mean means that in regions of the input space where there is no data, the predictions will be pushed to zero (which corresponds to the prior belief).

The covariance (or kernel) function $K(\cdot, \cdot) : \mathbb{R}^D \times \mathbb{R}^D \mapsto \mathbb{R}$ is a function that takes two regressors as arguments and returns a scalar value, which is a measure of the statistical correlation of those two points of the input space. In this case e.g. K_{ff} is an $N \times N$ matrix of covariances between each training data point.

If there is knowledge of the values $\boldsymbol{f}$, which correspond to the output values of the function $f(\cdot)$ from the training data in the case of no added noise, then it is possible to condition on these values to gain information on the function, which corresponds to doing standard

inference on a multi-variable normal distribution. After conditioning on the function points, the resulting probability distribution is called a posterior distribution, since it corresponds to confronting the prior beliefs about the function with the observed function values, and it encapsulates everything that is known about the function. The posterior is given by

$$p(\boldsymbol{f_*}|X_*, X, \boldsymbol{y}) \sim \mathcal{N}(\boldsymbol{\mu_*}, \Sigma_*) \tag{2-4}$$

$$\boldsymbol{\mu_*} = \boldsymbol{m_*} + K_{*f}K_{ff}^{-1}(\boldsymbol{y} - \boldsymbol{m_*}) = K_{*f}K_{ff}^{-1}\boldsymbol{y} \tag{2-5}$$

$$\Sigma_* = K_{**} - K_{*f}K_{ff}^{-1}K_{f*} \ . \tag{2-6}$$

In the case that the test data is corrupted by i.i.d. Gaussian noise, it is trivial to extend the inference procedure by simply adding the noise variance terms to the prior through an extra diagonal term (corresponding to i.i.d. Gaussian noise):

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f_*} \end{bmatrix} = \left(\begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}, \begin{bmatrix} K_{ff} + \sigma_n^2 I & K_{f*} \\ K_{*f} & K_{**} \end{bmatrix} \right) \tag{2-7}$$

and the posterior distribution then becomes

$$p(\boldsymbol{f_*}|X_*, X, \boldsymbol{y}) \sim \mathcal{N}(\boldsymbol{\mu_*}, \Sigma_*) \tag{2-8}$$

$$\boldsymbol{\mu_*} = \boldsymbol{m_*} + K_{*f}(K_{ff} + \sigma_n^2 I)^{-1}(\boldsymbol{y} - \boldsymbol{m_{f_*}}) = K_{*f}(K_{ff} + \sigma_n^2 I)^{-1}\boldsymbol{y} \tag{2-9}$$

$$\Sigma_* = K_{**} - K_{*f}(K_{ff} + \sigma_n^2 I)^{-1}K_{f*} \ . \tag{2-10}$$

The above distribution can be used to make predictions of a function value at a query point $\boldsymbol{x_*}$, by giving not only the statistical mean but also the variance of the prediction, which is a measure of the model's confidence about the prediction. The process of inference can be seen in Figure 2-1. The left-hand Figure shows the prior distribution, which is given by the assumptions taken before looking at the data. The mean function (which is not explicitly drawn) is zero valued over the entire domain, and the covariance function gives the variance of the predictions. The right-hand Figure shows the posterior distribution, where the functions are considered to pass through five noise-free observations. The posterior function values for the mean function (which is not explicitly drawn) and the 95% confidence bounds are given by Equations 2-9-2-10. The 95% confidence bounds are taken from the variance, corresponding to approximately twice the standard deviation of the predictor for any given query point. It is common to plot these as "error bars", and at each point the function value is estimated to be inside these error bars with high probability (in this case 95%).

Inference can be thought as drawing infinite samples from the prior distribution and then discarding all those that do not pass through the observed points (assuming noiseless observations), although in practice this is not how computations are carried out and Bayesian statistically conditioning on the observations allows to do this process in a trivial way.

GPR is a nonlinear, probabilistic and non-parametric Regression method:

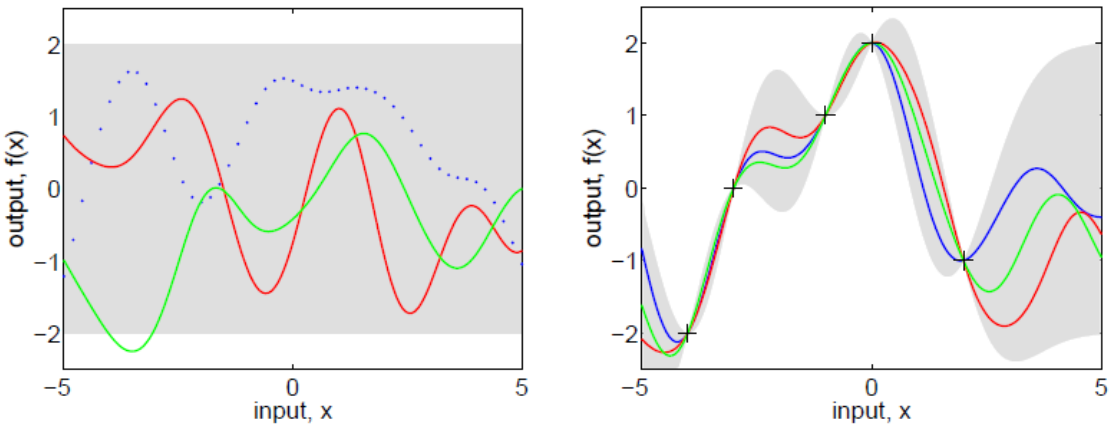

Figure 2-1: Process of inference in GPR a) Prior distribution and 3 sampled functions. b) Posterior distribution and 3 sampled functions. [18]

- Nonlinear since it is capable of mapping nonlinear input/output mappings.
- Probabilistic since the posterior is not a deterministic value, but a stochastic Gaussian distributed variable.
- Non-parametric since there is no need to choose one parameterized function model and optimize the parameters that maximize a certain functional, as it is done in other Identification methods. Instead, the covariance function specifies a family of functions by specifying how different target points are related to each other.

The conditioning on the training data is equivalent to keeping the functions that agree with the training data. GPR can be seen in a Bayesian framework as considering all the possible prior functions, and weighing them by how likely they explain the data set.

To see the difference from GPR to a parametric form of learning a function, consider Equation 2-9 rewritten as $\boldsymbol{\mu}(X_*) = \alpha \boldsymbol{y}$, with $\alpha = K_{*f} K_{ff}^{-1}$. Clearly the mean of the GP is linear in the target values $\boldsymbol{y}$ (in other words, the GP is a *linear smoother*), which shows that the information about the function is stored in the data points themselves and the training data is required every time the model needs to make a prediction. Conversely, a parametric method uses the data set *only* in the learning process to learn some parameters $\boldsymbol{\theta}$ that parameterize the function, i.e. $f(\boldsymbol{x}; \boldsymbol{\theta})$.

Another remark is that Equation 2-10 giving the variance of the prediction does not depend on the value of the targets $\boldsymbol{y}$, but only on the location of the regressors X. The variance term gives a measure of the data density around the prediction point; if there is a relatively high amount of training data around the query point the variance will be lower, and in sparsely trained areas the variance will be higher.
These last remarks should illustrate that the quality of the data set is especially important when using GPs.

The above described process of inferring a posterior distribution using GPR is simple to implement, and it is only required to pick the right prior for the Gaussian Process. Since the mean function is typically chosen to be zero, the covariance function is all that has to be specified by the user in practice, and the most important part of GPR. Insights on the covariance (kernel) function are provided in the next section.

2-2 Covariance function

One attractive property of Gaussian Processes is that the only information that has to be specified for practical implementations is the covariance function (or kernel function), which is a function describing the statistical relation between two points of the input space. The only restriction to picking a valid covariance function is that it needs to be a symmetric positive semi-definite function, that is, a function such that any subset of points X gives rise to a positive semi-definite matrix $K(X, X)$. A given kernel function can be proven to be positive semi-definite if it can be written as an inner product:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\varphi}(\boldsymbol{x}_i)^T \boldsymbol{\varphi}(\boldsymbol{x}_j) \quad . \tag{2-11}$$

In practice, there are multiple covariance functions that are known to be positive semi-definite, and the problem becomes choosing a covariance function that properly encapsulates the behaviour of the underlying function $f(\cdot)$. A poor choice of covariance function will result in a poor predictive model, since the model will be trying to fit the data set to a function space that does not contain functions qualitatively similar to $f(\cdot)$.

Fortunately, when it comes to choosing a kernel function there are a few popular choices that tend to give good results when applied to a variety of practical cases. Some of the most popular covariance functions will be introduced next:

2-2-1 ARD Squared Exponential kernel

The most popular choice by far is the Squared Exponential (SE) kernel

$$K_{SE}(\boldsymbol{x}_i, \boldsymbol{x}_j; \boldsymbol{\theta}) = \sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \Lambda^{-1} (\boldsymbol{x}_i - \boldsymbol{x}_j)\right) + \sigma_n^2 \delta_{ij} \quad . \tag{2-12}$$

This kernel function belongs to a special class of functions called stationary, where the value of the kernel only depends on the distance $\boldsymbol{x}_i - \boldsymbol{x}_j$ between the two points. If the points are close to each other, then the value of the kernel is high, and as the distance between points in the input space increases the covariance between the points decreases exponentially. A high value of the kernel means two points are highly correlated, and knowledge of the value $f(\boldsymbol{x}_i)$ gives relevant information on the value $f(\boldsymbol{x}_j)$. If the covariance between two points is low, then information on one point will not have a large impact on the prediction at the other point.

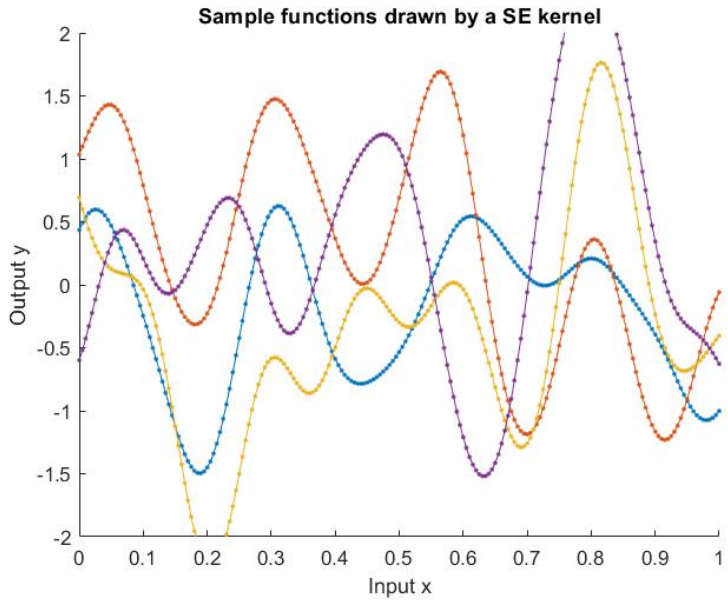

Figure 2-2: Sample functions drawn from a SE kernel

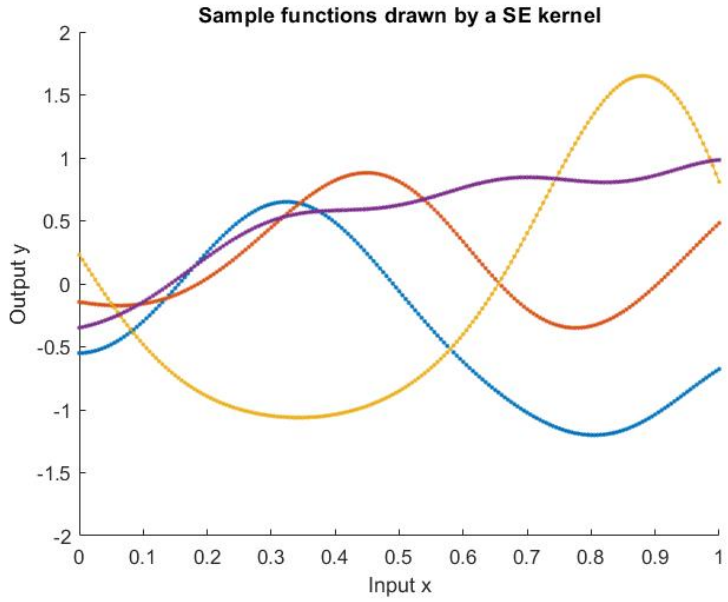

Figure 2-3: Sample functions drawn from similar a kernel to Figure 2-2 but with increased length-scale l_1

A few 1-dimensional functions sampled from a SE kernel are shown in Figure 2-2. These functions look smooth and are infinitely differentiable, which is caused by the fact that the kernel function is infinitely differentiable.

The kernel function is parameterized by a few variables, the so-called *hyper-parameters*, such

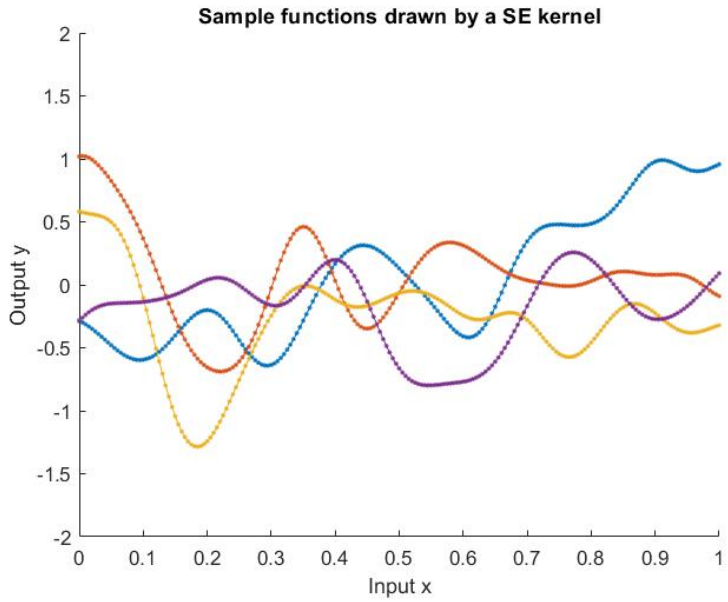

Figure 2-4: Sample functions drawn from similar a kernel to Figure 2-2 but with decreased variance σ_f

as σ_f, σ_n and $\Lambda^{-1} = \text{diag}\left(\left[l_1^{-2}, l_2^{-2}, ..., l_D^{-2}\right]\right)^2$. These hyper-parameters are tuning parameters, which can be jointly captured in a vector $\boldsymbol{\theta} = \begin{bmatrix}\sigma_f & \sigma_n & l_1 & ... & l_D\end{bmatrix}$, and changing their value will change the shape of the functions considered in the function space used for Regression. The hyper-parameter σ_f is a value representing the magnitude of deviation from the mean values. The hyper-parameter σ_n is the noise term in the measurements, and increasing its value corresponds to modelling more phenomena as noise. The hyper-parameter matrix $\Lambda^{-1} = \text{diag}\left([l_1^{-2}, l_2^{-2}, ..., l_D^{-2}]\right)$ is a matrix containing length-scales for all the input dimensions. Functions sampled from a similar kernel function as in Figure 2-2 but with different values for hyper-parameters can be seen in Figures 2-3 and 2-4.

A length-scale value l_i gives a measure of how much one has to move, for a given input dimension i, in order to get values that are not correlated. A large value of l_i gives rise to smooth and slow varying functions, whereas a low value of l_i gives rise to oscillatory and quickly varying functions. Since this kernel has a different length-scale for each individual dimension of the regressor, the kernel is said to have the property of Automatic Relevance Determination (ARD). Therefore Equation 2-12 is an ARD SE kernel function.

Along with the choice of covariance function, the choice of hyper-parameters is crucial to the success of GPR, and these hyper-parameters are usually chosen from an optimization problem, which will be considered in the next section.

[2] $\Lambda^{-1} = \text{diag}\left(\left[l_1^{-2}, l_2^{-2}, ..., l_D^{-2}\right]\right)$ stands for a square matrix whose diagonal elements are the function argument and the non-diagonal entries are zero.

2-2-2 Linear kernel

A linear kernel has the following form:

$$K_{Lin}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_o^2 + \boldsymbol{x}_i^T \Lambda^{-1} \boldsymbol{x}_j \tag{2-13}$$

where the hyper-parameters are $\boldsymbol{\theta} = \begin{bmatrix} \sigma_o & l_1 & ... & l_D \end{bmatrix}$. The linear kernel gives rise to functions which are affine in the inputs. σ_o is an offset term and $\Lambda = \text{diag}\left([l_1^{-2}, l_2^{-2}, ..., l_D^{-2}]\right)$ represents the length-scales which give the derivatives in each input direction. GPR with a Linear kernel can be shown to be equivalent to a form of Linear Ridge Regression [20]. A few 1-dimensional functions sampled from a linear kernel are shown in Figure 2-5.

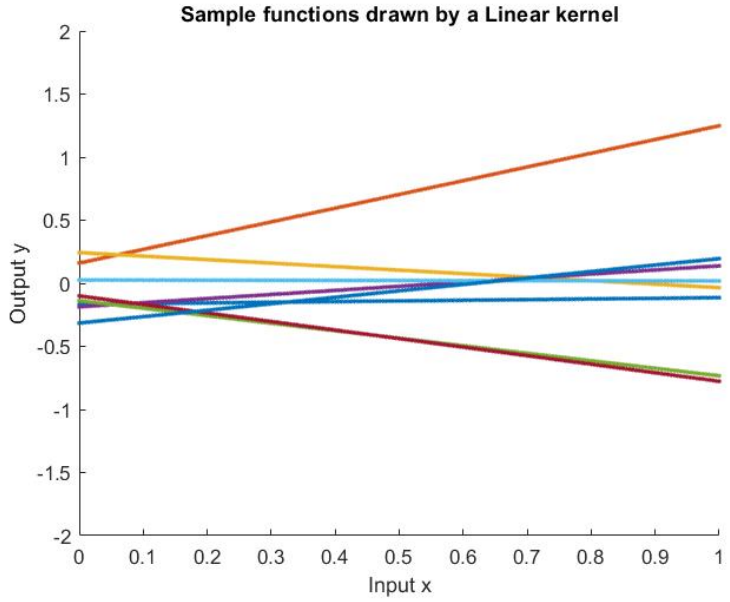

Figure 2-5: Sample functions drawn from a linear kernel

Figures 2-6 and 2-7 show the effects of changing the kernel hyper-parameters. Increasing σ_o will increase the variance of the offset value, leading the GP to consider functions with a larger constant term. Increasing the length scales l_i means considering functions with higher derivative values along the input dimension i.

2-2-3 Periodic kernel

Often times a function will exhibit periodic behaviour, and the Periodic kernel can be used to account for such behaviour. One example of a Periodic kernel is the following

$$K_{Per}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{2\sin^2\left(\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|}{2}\right)}{l^2}\right) \tag{2-14}$$

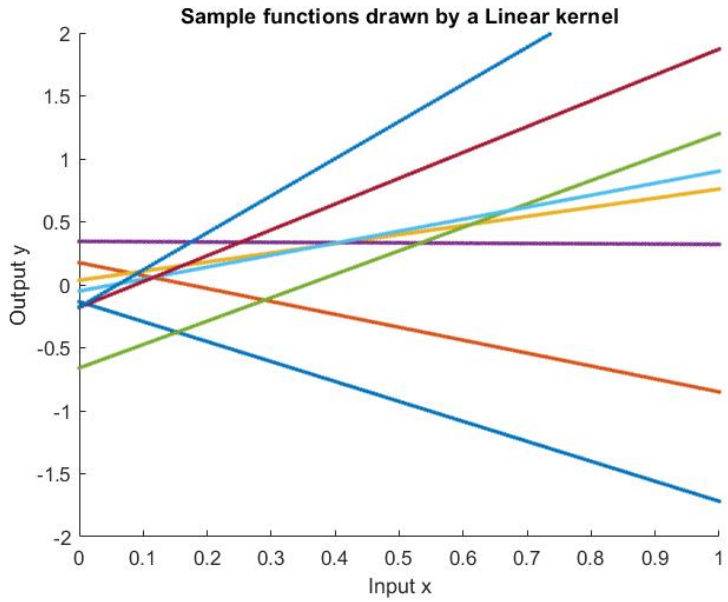

Figure 2-6: Sample functions drawn from a kernel similar to Figure 2-5 but with increased length-scale l_1

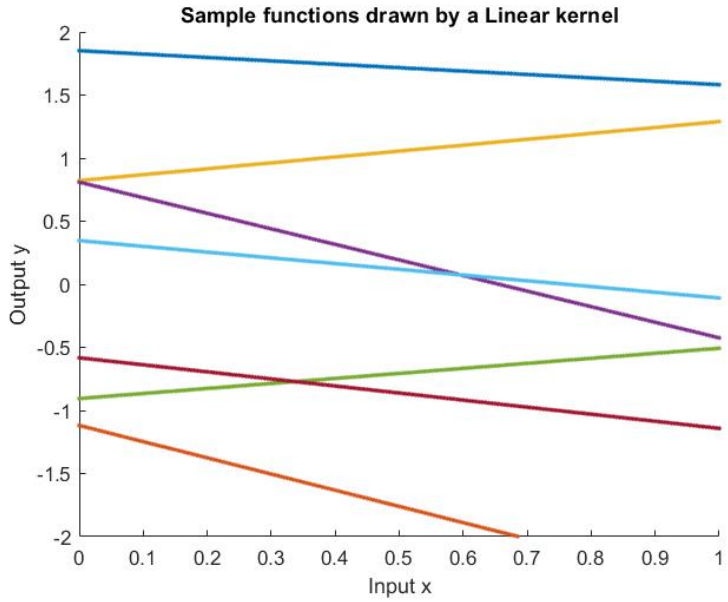

Figure 2-7: Sample functions drawn from a kernel similar to Figure 2-5 but with increased offset σ_o

where the hyper-parameter(s) is the length scale l giving the period of the considered family of functions. Functions sampled from this kernel with similar length-scales are shown in Figure 2-8, all of which show the desired repeating pattern along the input space.

Periodic phenomena happens in a lot of physical systems, often in conjunction with other trends from previously seen kernels. For example, a dynamic mass-spring-damper system

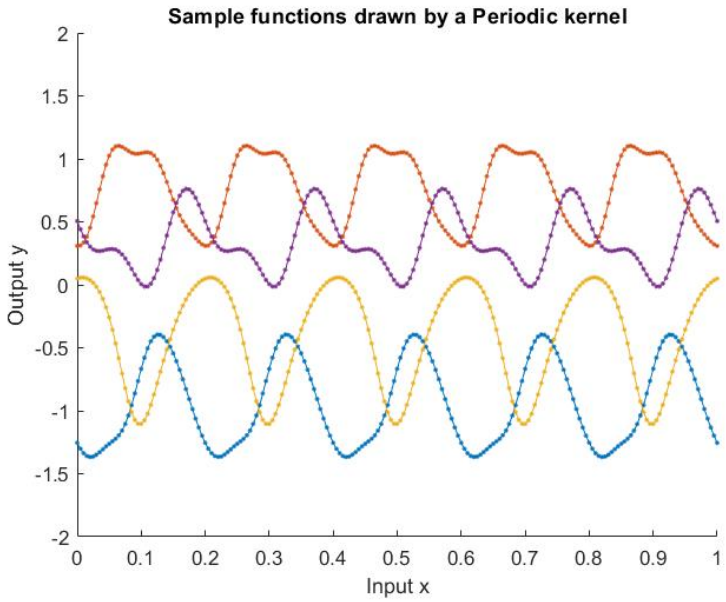

Figure 2-8: Sample functions drawn from a Periodic kernel

shows stable under-damped oscillations that can be modelled by a periodic function wrapped by an exponentially decaying function. These types of trends can be captured for GPR by combining the previous kernels and making new ones, which is briefly considered next.

2-2-4 Operations on kernels

Certain operations over kernel functions always result in another valid kernel function. Since a kernel function has to yield a symmetric positive semi-definite kernel matrix for any combination of input points, any operation that maintains this property can be used to obtain new kernels (it is said that a kernel is *closed* under these operations). The most common of these operations are listed bellow:

- sum: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = K_1(\boldsymbol{x}_i, \boldsymbol{x}_j) + K_2(\boldsymbol{x}_i, \boldsymbol{x}_j)$.
- product: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = K_1(\boldsymbol{x}_i, \boldsymbol{x}_j) K_2(\boldsymbol{x}_i, \boldsymbol{x}_j)$.
- re-scaling: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = a K_1(\boldsymbol{x}_i, \boldsymbol{x}_j)$.
- multiplication by a function: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = f(\boldsymbol{x}_i) K_1(\boldsymbol{x}_i, \boldsymbol{x}_j) f(\boldsymbol{x}_j)$.
- composition with a function: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = K_1(f(\boldsymbol{x}_i), f(\boldsymbol{x}_i))$.

These operations can be used to further impose structure in the function being estimated. For example, to estimate the function representing the dynamics of the previously mentioned mass-spring-damper system, it makes sense to use a product of a Periodic kernel, representing the oscillatory behaviour, with a SE kernel, representing the damped oscillations, i.e. $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = K_{SE}(\boldsymbol{x}_i, \boldsymbol{x}_j) K_{Per}(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

2-3 Choice of kernel hyper-parameters

The last section showed that most of the information about the space of functions used for Regression is encoded in the covariance function. Not only is the choice of covariance function important, but also the choice of hyper-parameters associated with it. The name hyper-parameter is meant to emphasize that GPR is not a parametric Regression model, since it does not make use of a specific parameterized function to make predictions. Instead, the hyper-parameters can be viewed as free values in the covariance function which specify a family of functions to consider at inference time. This Section explains how to optimize the choice of hyper-parameters, by first considering a Bayesian view of model selection.

GPR is closely related to a Bayesian probability framework, where instead of considering a single function one considers a probability distribution that averages over all possible functions. The Bayesian framework of model selection can be carried out at different levels. In the first level inference is carried out to obtain a posterior. The posterior distribution $p(\boldsymbol{f}|X,\boldsymbol{y},\boldsymbol{\theta},H_i)$ is a probability distribution over possible functions. Here, $\boldsymbol{\theta}$ represents the hyper-parameters and H_i represents a kernel function picked from a discrete set of possible kernel functions under consideration. At the lower level, the hyper-parameters and the covariance function are assumed to be chosen, and inference is carried out using Bayes' rule by taking a prior $p(\boldsymbol{f}|\boldsymbol{\theta},H_i)$, which encodes prior beliefs about the function, and a likelihood $p(\boldsymbol{y}|X,\boldsymbol{f},\boldsymbol{\theta})$, which encodes how likely it is that a function explains the data. Inference takes the form

$$p(\boldsymbol{f}|X,\boldsymbol{y},\boldsymbol{\theta},H_i) = \frac{p(\boldsymbol{y}|X,\boldsymbol{f},\boldsymbol{\theta})p(\boldsymbol{f}|\boldsymbol{\theta},H_i)}{p(\boldsymbol{y}|X,\boldsymbol{\theta},H_i)} \quad . \tag{2-15}$$

A function $f(\cdot)$ will be assigned a high probability density if it agrees with the prior beliefs and if it explains the data well. The term in the denominator $p(\boldsymbol{y}|X,\boldsymbol{\theta},H_i)$ is called the marginal likelihood, and it is an important quantity for reasons that will be explained shortly.

Choosing a model also implies choosing proper hyper-parameters. Continuing to follow the Bayesian formalism, instead of simply choosing specific values for the parameters $\boldsymbol{\theta}$ one should consider all possible parameters $\boldsymbol{\theta}$ and assign a probability distribution over them as well. This is the second level of inference. Similarly as before, using Bayes' rule

$$p(\boldsymbol{\theta}|X,\boldsymbol{y},H_i) = \frac{p(\boldsymbol{y}|X,\boldsymbol{\theta},H_i)p(\boldsymbol{\theta}|H_i)}{p(\boldsymbol{y}|X,H_i)} \quad . \tag{2-16}$$

This time the likelihood term for the hyper-parameters, which corresponds to how likely it is that a given choice of hyper-parameters explains the data, is the marginal likelihood from the first level of inference. The term $p(\boldsymbol{\theta}|H_i)$ would be a prior specified over the hyper-parameters (hyper-prior), corresponding to beliefs about these values, and the denominator $p(\boldsymbol{y}|X,H_i)$ is the normalizing constant

$$p(\boldsymbol{y}|X,H_i) = \int p(\boldsymbol{y}|X,\boldsymbol{\theta},H_i)p(\boldsymbol{\theta}|H_i)d\boldsymbol{\theta} \quad . \tag{2-17}$$

Equation 2-16 gives a probability distribution over the hyper-parameters.
Continuing with the Bayesian formalism, there is no need to simply choose one kernel function. Instead, one should use a probability distribution over the set of our possible kernel choices, again according to Bayes' rule

$$p(H_i|X,\boldsymbol{y}) = \frac{p(\boldsymbol{y}|X,H_i)p(H_i)}{p(\boldsymbol{y}|X)} \tag{2-18}$$

where again $p(H_i)$ is a prior over the discrete set of covariance functions, representing prior beliefs, and $p(\boldsymbol{y}|X)$ is a normalizing constant

$$p(\boldsymbol{y}|X) = \sum_i p(\boldsymbol{y}|X,H_i)p(H_i) \ . \tag{2-19}$$

What is described above is the ideal Bayesian treatment for choosing a model. Although it is very elegant, in reality it is impossible to implement because the integrals in the denominator become intractable. Specifically, in the second level of inference in Equation 2-16 calculation of $p(\boldsymbol{y}|X,H_i)$ is intractable for most cases. This makes it so that the posterior over the hyper-parameters cannot be calculated.

One way to get around this problem can be to use Monte Carlo sampling methods to approximate the integral. However, this solution is computationally slow and cumbersome. The most popular option is to part ways with the fully Bayesian model selection and make some optimal point choice of hyper-parameters $\hat{\boldsymbol{\theta}}$ that maximize an optimization criterion. This proves to be a good enough approximation if the distribution $p(\boldsymbol{\theta}|X,H_i)$ is "well peaked", meaning that the model is fairly certain about the value of the hyper-parameters. In fact, considering a specific value $\hat{\boldsymbol{\theta}}$ is the same as considering $p(\boldsymbol{\theta}|X,H_i)$ as a Dirac distribution centered in $\hat{\boldsymbol{\theta}}$, which would mean that the model has absolute certainty of the hyper-parameters. The distribution $p(\boldsymbol{\theta}|X,H_i)$ is often more peaked the more training data is considered, since fewer values of $\boldsymbol{\theta}$ are able to reasonably explain the data.

The question becomes how to determine the hyper-parameters $\hat{\boldsymbol{\theta}}$. The goal is to find the point maximum of $p(\boldsymbol{\theta}|X,H_i)$, which up to a normalizing constant can be written (from Equation 2-16) as

$$p(\boldsymbol{\theta}|X,\boldsymbol{y},H_i) \propto p(\boldsymbol{y}|X,\boldsymbol{\theta},H_i)p(\boldsymbol{\theta}|H_i) \ . \tag{2-20}$$

If the prior on the hyper-parameters is flat (which corresponds to saying that there is no prior belief on the value of $\boldsymbol{\theta}$ and therefore every value is equally likely), then the distribution becomes proportional to the marginal likelihood, which can be used as a functional depending on the hyper-parameters and solved as an optimization problem. For mathematical convenience, the log likelihood is taken instead

$$F(\boldsymbol{\theta}) = \log p(\boldsymbol{y}|X,\boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}^T K_y^{-1}\boldsymbol{y} - \frac{1}{2}\log|K_y| - \frac{n}{2}\log 2\pi \tag{2-21}$$

where the dependency on H_i was dropped out (the choice of covariance function is left to the user). The choice of kernel hyper-parameters to be used for inference will be

$$\hat{\boldsymbol{\theta}} = \text{argmax } F(\boldsymbol{\theta}) \ . \tag{2-22}$$

Analyzing the individual terms of $F(\boldsymbol{\theta})$ can give insight into the choice of hyper-parameters. The first term is the only one depending on the training target values $\boldsymbol{y}$, and it promotes data-fitting. The second term containing the norm of the kernel matrix is a penalty term on the complexity of the model. The third term is just a normalizing constant, irrelevant for the optimization problem. The inclusion of both a data-fit term and a penalty term gives a trade-off (the so-called *Occam's razor*) between the desire to obtain a model that accurately predicts the training data and the desire to obtain a simple model in order to avoid over-fitting. Similarities between GPR and Ridge Regression are discussed in [18].

Obtaining a value of $\hat{\boldsymbol{\theta}}$ is possible by solving the above optimization problem. This can be achieved with standard optimization algorithms such as gradient descent, if one has access to the partial derivatives of the log-likelihood. It is a well known result that the partial derivatives can be calculated for a generic one-time differentiable kernel function with

$$\frac{\partial}{\partial \theta_j} \log p(\boldsymbol{y}|X, \boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{y}^T K^{-1} \frac{\partial K}{\partial \theta_j} K^{-1}\boldsymbol{y} - \frac{1}{2}\text{tr}\left(K^{-1}\frac{\partial K}{\partial \theta_j}\right) \tag{2-23}$$

where $\text{tr}(\cdot)$ stands for the trace of a matrix. Using the above expressions in combination with a gradient-based optimization algorithm gives the estimates $\hat{\boldsymbol{\theta}}$, and performing inference on the training data (see Section 2-1) with the covariance function $k(\boldsymbol{x}_i, \boldsymbol{x}_j; \boldsymbol{\theta})$ gives the desired GP posterior distribution.

2-4 Model Validation

Validation is an important part of Regression (and System Identification), since it gives a measurement of how well the obtained model agrees with the behaviour of the underlying function (or process). The validation procedure should not be done using the training data, since that does not give information about over-fitting. Instead, it is common to separate the data set at the beginning into two sets: one used for training (containing most of the points) and one used for validating the model (containing a small subset of data points). This is often called cross-validation, and it is common practice to all System Identification techniques. Often the data is divided into k non-overlapping subsets, one of the k subsets is used for validation and all other sets are used for training. The validation procedure is then repeated k times by rotating the set that is used for validation, and the model performance is considered as an average of all evaluations.

What is left is to choose a measure that allows to compare the data. Since a GP is a probabilistic model it gives a mean as well as a variance prediction, and it opens more possibilities for potential loss functions to be use in this validation step, some of which are presented next.

The most common loss functions used for validation are the Mean Squared Error (MSE) and the Standardized Mean Squared Error (SMSE)

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mu_i)^2 \tag{2-24}$$

$$\text{SMSE} = \frac{1}{N}\frac{\sum_{i=1}^{N}(y_i - \mu_i)^2}{\sigma_y^2} \tag{2-25}$$

using the notation σ_y as the variance of the targets, y_i as the output measured from the real system and $\hat{y}_i$ as the predicted output distribution, with mean and covariance values $\hat{y}_i \sim \mathcal{N}(\mu_i, \sigma_i)$. While these are perfectly acceptable validation criteria, it is worth considering other methods that include the predicted variance when judging the quality of the models. The Log Density Error (LDE) (used in [21]) is one such method:

$$\text{LDE} = \frac{1}{2N}\sum_{i=1}^{N}\left(\ln(\sigma_i^2) + \frac{(y_i - \mu_i)^2}{\sigma_i^2}\right) \quad . \tag{2-26}$$

The LDE validation method takes into account the variance in two distinct ways: the first term inside the sum penalizes high variance, negatively valuing predictions that are uncertain; and the second term of the sum is weighted by σ_i^{-1}, penalizing the error in predictions that have overconfident variance terms. This is important since it is not always ideal to have a small variance term: if the model is uncertain in the prediction then it is desirable that the variance term is large, and making wrong predictions with small variance shows a poor performance from the model.
A similar metric to LDE is the Mean Standardized Log Loss (MSLL) (used in [18]):

$$\text{MSLL} = \frac{1}{2N}\sum_{i=1}^{N}\left(\ln(\sigma_i^2) + \frac{(y_i - \mu_i)^2}{\sigma_i^2}\right) - \frac{1}{2N}\sum_{i=1}^{N}\left(\ln(\sigma_y^2) + \frac{(y_i - \bar{y})^2}{\sigma_y^2}\right) \tag{2-27}$$

using the notation $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$. This loss function takes the LDE loss and subtracts the loss of a trivial predictor that simply uses the mean and variance values of the targets. The advantage is that the MSLL gives more interpretable results, as a simpler model will have close to zero loss values and a better model will have negative loss values.

Different loss functions can be considered based on different criteria. For instance, Akieke's Information Criterion (AIC), coming from the field of information theory, is given by

$$\text{AIC} = -2F(\hat{\boldsymbol{\theta}}) + 2\alpha \quad . \tag{2-28}$$

AIC makes use of the negative log likelihood $F(\hat{\boldsymbol{\theta}})$ (from Equation 2-21), which gives the likelihood that the data is explained by the model. It also has a term α, which is the number of tuning parameters in the model. This is a simple way to try and penalize over-fitting by

penalizing the use of a lot of kernel hyper-parameters.

Another possible model validation strategy is residual analysis, using metrics such as correlation tests, whiteness tests, etc. In practice, using a loss function that concerns the mean values, such as MSE or SMSE, in combination with a loss function that takes the variance term into account, such as MSLL, and considering both results is a simple and effective way of performing the validation step.

2-5 Why use Gaussian Process Regression?

GPR has had a lot of developments in the past few decades. Its application in Regression and Classification problems started to gain attention after it was shown that GPs could outperform Neural Networks in several Regression tasks [11].

Its advantages over different nonlinear modelling techniques are: the fact that the probabilistic solution gives a measure of the uncertainty, in the form of variance; the fact that it is a non-parametric model with a relatively low amount of hyper-parameters to optimize (which is especially useful for high dimensional input spaces); the simplicity of implementation and elegant framework; and the fact that it can be considered as a universal function approximator [6]. In fact, GPR can be viewed as doing Linear Regression with a set of nonlinear basis functions, and kernels such as the SE kernel can be shown to be equivalent to considering an infinite set of basis functions (this is called a non-degenerate kernel) [18].

Several methods of Regression can be thought of as special cases of GP, such as Radial Basis Functions, Splines, Linear Ridge Regression, or Neural Networks with one hidden layer. In fact, GPR with a SE kernel can be interpreted as a Neural Network with a single layer and an infinite number of neurons given by Radial Basis Functions (see [22]).

All these reasons contribute to the popularity of GP methods. In the context of this thesis, using GPs for System Identification will provide a model that gives "extra" information on the quality of the prediction through the variance term, which can be used for control applications. Not only that, but the non-parametric structure of the GP means that the information is directly stored in the data points, which always have to be accessed to make predictions. This is an indication that it might be reasonable to use GP models in Data-driven control applications. However, there are disadvantages in using it, the biggest of which is the computational complexity that scales cubically in the number of points, i.e. $\mathcal{O}(N^3)$. This prevents GPs to be realistically employed in applications where the number of data points is larger than a few thousand, which is a big limitation of the method.

The next chapter is dedicated to implementing GPR as a means for System Identification, as well as addressing some of the challenges of implementation to applying GPR to a time-series dynamic model.

Chapter 3

Gaussian Processes for System Identification

Section 2 introduced Gaussian Process Regression (GPR). The goal of this section is to apply the methods of a Gaussian Process (GP) to the learning and Identification of dynamic systems.

Let us start by discussing the advantages of using GPs for System Identification. Most System Identification frameworks deal with linear systems, since linear systems are simpler to work with as they can be studied from very established knowledge of Linear Algebra. In many applications the system exhibits behaviour linear enough that a linear system can accurately model it. However, systems with important nonlinear behaviour are also common, and non-linear Identification techniques have to be applied. This thesis focuses on the use of GPs to learn this (possibly) nonlinear unknown function representing the system dynamics.

A GP has properties that make it an interesting candidate model. It is a non-parametric model, which means that there is no fixed set of parameters that have to be learned and then used to map to a specific function. GPR uses a Bayesian framework to integrate out the parameters, such that a space of possible functions is considered, and the model averages these functions by how likely they describe the data. Information about the function space is introduced through the covariance function, which still can have hyper-parameters to be learned. However, this is different from learning parameters in a parametric model, where a specific structure is imposed in the Identification phase. The practical consequences of this are that GPs have a relatively lower number of (hyper-)parameters to learn, an easy way of incorporating prior knowledge into the Identification process, and a natural solution to over-fitting, since they average through all the possible functions and they consider an over-fitting term in the process of kernel optimization (see Section 2-3). They also impose few restrictions to the structure of the function, as the assumption of a joint Gaussian distribution between data points can capture the behaviour of many functions [6].

Another advantage of using GPs is that the model is probabilistic in nature, which means that for a certain prediction we get information on not only the next output value, but also the variance of this prediction. This will give information about the confidence of the prediction. This variance comes naturally from the normal distribution of the output, and this extra information is unique to this method.

There are also disadvantages of modelling with GPs. One disadvantage is that the method is computationally complex and scales with $\mathcal{O}(N^3)$. This problem has been addressed extensively in the literature and will be discussed in Chapter 4. Another disadvantage is that a good model can only be obtained in regions of the input space where there is enough available data. GPs are typically bad extrapolators, and data covering all regions of the input space is necessary. The reason why this happens is that for a non-parametric model the information is stored in the learning data itself, since the outputs of the data set are always used at prediction time. A low density of data for a certain input region typically means that predictions at that region will be poor, since the prediction will be dominated by the prior and this prior is often picked frivolously. However, even when the model prediction is poor we can know about it through the variance term in the predictions.

This last point is especially interesting and unique for GPs. Information about the variance of the model can be used in simulations and control schemes in a unique way that other methods do not allow. For example, [23] used a GP in Identification to detect seizures in newborn babies through the use of the variance term. The idea was to measure electric signals in newborns' brains using an EletroEncephaloGram (EEG) test and learn a GP model from the time data, since the occurrence of a seizure would cause an increase in variance.

The variance in GPs can be used in many other interesting ways. It can be used as a constraint in Predictive Control, or as a penalty to the System's inputs. The logic is that if there is a lot of uncertainty about the model, more conservative control strategies should be employed. Another use for GPs is for online Data-driven applications. The variance term can be used to determine if a certain input region has sufficient data, and less explored regions can be prioritized for exploration based on this criteria.

This Chapter introduces the use of GPs for System Identification. Section 3-1 discusses the adaptation of the GP structure to input/output data from the System. Section 3-2 discusses different methods to perform multiple-steps-ahead predictions with probabilistic models. Finally, Section 3-3 discusses adaptations to online implementations.

3-1 Model structures and choice of regressors

There are several different model structures to be considered for GP Identification. GPR tries to learn the mapping of a set of regressors to a set of targets corrupted by a noise term

$$y = f(\boldsymbol{x}) + \epsilon \tag{3-1}$$

where ϵ is i.i.d. Gaussian distributed noise $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. For the Identification problem we often have a data set $\{u_i, y_i\}_{i=1}^N$, where u_i is an input at a certain time step i and y_i is the corresponding output. For Systems with multiple outputs one only needs to learn a GP for each of the output dimensions. Since we will be analysing time series data, there is a dependency on the time step k

$$y_k = f(\boldsymbol{x}_k) + \epsilon_k \quad . \tag{3-2}$$

The choice of the regressors is important and it determines the model structure. If we consider a Nonlinear AutoRegressive eXogenous (NARX) model (see Figure 3-1), then the regressors are a past window of inputs and outputs of the System

$$\boldsymbol{x_k} = [y_{k-1}, y_{k-2}, ..., y_{k-n}, u_{k-1}, u_{k-2}, ..., u_{k-m}]^T \quad . \tag{3-3}$$

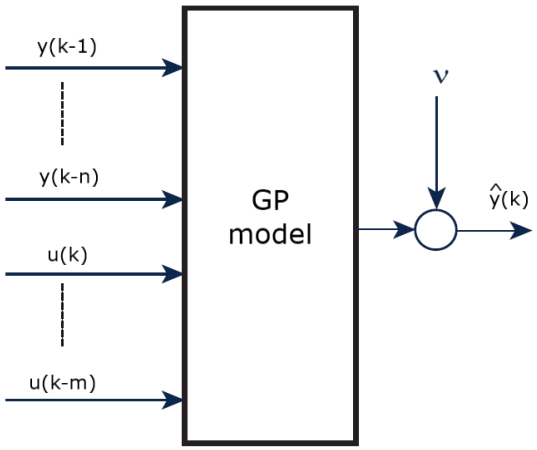

Figure 3-1: Diagram of the GP-NARX model structure

A set of data matrices $\{X_i, \boldsymbol{y_i}\}_{i=1}^N$ can be obtained for Regression by stacking these regressors and the System output as

$$X = [\boldsymbol{x}_1 \boldsymbol{x}_2, ..., \boldsymbol{x}_N] \in \mathbb{R}^{(n+m)\times N} \tag{3-4}$$

$$\boldsymbol{y} = [y_k y_{k+1}, ..., y_{k+N}]^T \in \mathbb{R}^{N\times 1} \tag{3-5}$$

and the structure of a NARX-GP model then becomes

$$y_k = f(y_{k-1}, y_{k-2}, ..., y_{k-n}, u_{k-1}, u_{k-2}, ..., u_{k-m}) + \epsilon_k \tag{3-6}$$

where the goal is to learn the function $f(\cdot)$ modeled by a Gaussian Process. Particular forms of the NARX-GP model are the Nonliner Finite Impulse Response (NFIR) model, which only considers a past window of inputs (alternatively, $n = 0$)

$$y_k = f(u_{k-1}, u_{k-2}, ..., u_{k-m}) + \epsilon_k \tag{3-7}$$

and the Nonlinear AutoRegressive (NAR) model, which only considers a past window of outputs

$$y_k = f(y_{k-1}, y_{k-2}, ..., y_{k-n}) + \epsilon_k \quad . \tag{3-8}$$

It is also possible to learn a State-Space representation of the System, the so-called Gaussian Process State Space Model (SSM). The most general representation of a GP-SSM is the following

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, u_k) + \boldsymbol{\epsilon}_k^{(1)} \tag{3-9}$$

$$y_k = g(\boldsymbol{x_k}, u_k) + \epsilon_k^{(2)} \tag{3-10}$$

Where now both System dynamics $f(\cdot)$ and observer dynamics $g(\cdot)$ have to be identified. In this GP-SSM model structure the Regression process is not so simple, since there is no access or knowledge of the latent states $\boldsymbol{x}_k$. The Bayesian framework can allow to integrate out the latent states, using techniques such as Markov Chain Monte Carlo (MCMC) Sampling methods [24] or Variational Inference methods [25]. While these methods have received a lot of attention in the last years, they are outside of the scope of this thesis. Instead this work will consider the assumption that there is direct access to the state (regressor) values:

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, u_k) + \boldsymbol{\epsilon}_k^{(1)} \quad . \tag{3-11}$$

This simplifying assumption allows to use the Regression methods from Chapter 2, since now it's possible to collect input/output data for training.

An important question is which of the above models to use for a given application. To answer this question, let's further consider the choices one has to make for each model structure. Considering again the NARX model, selecting the regressors is an important step, since after deciding the past window of input/outputs to use we can use the methods from Chapter 2 to learn the dynamics $f(\cdot)$. The choice of regressors can be done in several ways. One way is to use a brute force approach, which means that different models with different regressors can be learned and then compared in some later validation step. A second way of learning regressors is by studying the statistical properties of the data set, such as correlation, auto-correlation, etc. A third interesting way of learning relevant regressors exists in GPs by exploring the structure of certain covariance functions and hyper-parameter learning and using Automatic Relevance Determination (ARD) kernels (see Section 2-2-1). For example, consider the very common Squared Exponential (SE) covariance function

$$C_f(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_f^2 \exp\left[-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \Lambda^{-1} (\boldsymbol{x}_i - \boldsymbol{x}_j)\right] + \delta_{ij}\sigma_n^2 \tag{3-12}$$

where the term $\Lambda = \text{diag}([l_1^{-2}, ..., l_d^{-2}])$ is a diagonal matrix with the characteristic length scales of every dimension in the regressor vector. These are the hyper-parameters learned by the GP through the optimization of the log marginal likelihood (see Section 2-3). If a length scale is large, that means a large change in that specific regressor value is necessary to significantly change the function output. That might lead to the conclusion that the specific regressor is not a relevant input and can be removed from the model.

For example, consider the modeling of a nonlinear dynamical System depending only on the input and output at the previous time step

$$y_k = f^{\text{true}}(y_{k-1}, u_{k-1}) + \epsilon_k \quad . \tag{3-13}$$

If a model is learned using the regressor vector $\boldsymbol{x}_k = [y_{k-1}, y_{k-2}, u_{k-1}, u_{k-2}]^T$, then after the hyper-parameter optimization step one should find that the length scales l_2 and l_4 corresponding to y_{k-2} and u_{k-2} are large, and simplify the model by removing these regressors.
This is an advantage of the NARX model, since this model does not have to make any assumptions on the order of the System. Instead, a sufficiently large window of past inputs and outputs should be considered, and the use of ARD will help to select only a window of regressors that add non-redundant information about the function, with the goal of considering as little regressors as possible. This allows the NARX model to be applied to Systems with very little prior knowledge of the underlying dynamics.

On the other hand it should be noted that this past window of inputs and outputs are heavily correlated, i.e. inputs such as y_{k-1} and y_{k-2} share a lot of the same information, which leads to having a larger input dimension than we would like. This is an advantage of State-Space Models, which determine the states that are relevant to the System's dynamics and therefore deal with smaller dimension input spaces. In general, it is a good policy to combine NARX modelling with expert knowledge when choosing the proper data set and regressor structure to use for training.

3-2 Making multiple-step-ahead predictions

The problem of dealing with uncertain Gaussian inputs for doing prediction in GPs was dealt with by [26], [27] and is especially relevant for learning dynamic systems or dealing with time series data. This is not only because the system output may be corrupted by sensor noise, which can in most cases be modelled by an i.i.d. Gaussian, but also because it is necessary to make multiple-steps-ahead predictions. This can be seen by considering a NARX model which is used to make a prediction at a time step k

$$\tilde{y}_k = f(y_{k-1}, y_{k-2}, ..., y_{k-n}, u_{k-1}, u_{k-2}, ..., u_{k-m}) + \epsilon_k \quad . \tag{3-14}$$

The one-step-ahead prediction $\tilde{y}(k)$ is a probabilistic Gaussian distribution with a mean value $\hat{y}_k$ and a variance σ_k, i.e. $\tilde{y}_k = \mathcal{N}(\hat{y}_k, \sigma_k)$. If we wish to make a two-step-ahead prediction, then the variable $\tilde{y}_k$ becomes an input to the system.

$$y_{k+1} = f(\tilde{y}_k, y_{k-1}, ..., y_{k+1-n}, u_k, u_{k-1}, ..., u_{k+1-m}) + \epsilon_k \quad . \tag{3-15}$$

So far, only deterministic inputs have been considered. For the above mentioned reasons, its important to consider solutions to dealing with noisy inputs given by normal distributed stochastic regressors. The problem takes the following mathematical representation

$$y = f(\boldsymbol{x}) + \epsilon_y \tag{3-16}$$
$$\boldsymbol{x} \sim \mathcal{N}(\hat{\boldsymbol{x}}, \Sigma_x) \quad . \tag{3-17}$$

There are three main alternatives that will be introduced next, as well as a comparison between them in terms of accuracy and simplicity.

a) Naive Approach: The naive approach consists of considering only the mean value $\hat{\boldsymbol{x}}$ for the next step prediction. Essentially the prediction takes the most likely value of the output at the previous step as certain and ignores the variance, that is

$$y \approx f(\hat{\boldsymbol{x}}) + \epsilon_y \quad . \tag{3-18}$$

The naive approach is the simplest solution, and its recommended if one is only interested in the mean prediction values. However, to take advantage of the confidence intervals of the GP model and get reasonable variances at each time step it is required to account for the input uncertainty, especially as the prediction horizon increases.

If the variance of the inputs is taken into account, then the prediction of y needs to be evaluated over all possible values of $\boldsymbol{x}$

$$p(y|\boldsymbol{x}) = p(y|\hat{\boldsymbol{x}}, \Sigma_x) = \int p(y|\boldsymbol{x}) p(\boldsymbol{x}|\hat{\boldsymbol{x}}, \Sigma_x) d\boldsymbol{x} \quad . \tag{3-19}$$

The result from the integral in Equation 3-19 is no longer a normal distribution. Instead, $p(y|\boldsymbol{x})$ can become arbitrarily complex and intractable (see Figure 3-2). To recover a Gaussian, it's necessary to make an approximation of this true distribution. One possible way of doing this is to sample values from the input distribution $\boldsymbol{x}$ and approximate the integral with Monte-Carlo methods, but this is not the most interesting solution because it's very computationally demanding. The alternative is to approximate the integrals by computing only the first and second statistical moments

$$p(\boldsymbol{y}|\hat{\boldsymbol{x}}, \Sigma_x) \approx \mathcal{N}(m(\hat{\boldsymbol{x}}, \Sigma_x), \sigma(\hat{\boldsymbol{x}}, \Sigma_x)) \tag{3-20}$$

Here, the notation from [26] is adopted, where $\mu(\hat{\boldsymbol{x}})$ and $\sigma^2(\hat{\boldsymbol{x}})$ are the Gaussian mean and covariance for a noise free deterministic input value $\hat{\boldsymbol{x}}$, while $m(\hat{\boldsymbol{x}}, \Sigma_x)$ and $v(\hat{\boldsymbol{x}}, \Sigma_x)$ are the Gaussian mean and covariance for an approximation of the distribution with noisy inputs.

It has been shown [26] that expressions for the approximate Gaussian moments can be given by

$$m(\hat{\boldsymbol{x}}, \Sigma_x) = \sum_{i=1}^{N} \beta_i \mathbb{E}[K(\boldsymbol{x}, \boldsymbol{x}_i)] \tag{3-21}$$

$$v(\hat{\boldsymbol{x}}, \Sigma_x) = \mathbb{E}[K(\boldsymbol{x}, \boldsymbol{x})] - \sum_{i,j=1}^{N} (K_{i,j}^{-1} - \beta_i \beta_j) \mathbb{E}[K(\boldsymbol{x}, \boldsymbol{x}_i) K(\boldsymbol{x}, \boldsymbol{x}_j)] - m(\hat{\boldsymbol{x}}, \Sigma_x)^2 \tag{3-22}$$

Where the term β_i stands for row i of the vector $\boldsymbol{\beta} = (K_{ff} + \sigma_n^2 I)^{-1} \boldsymbol{y}$. The above expressions present the approximate Gaussian posterior distribution as a function of a few covariance terms depending on $\boldsymbol{x}$. These terms are the following:

$$\mathbb{E}[K(\boldsymbol{x}, \boldsymbol{x})] = \int K(\boldsymbol{x}, \boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x} \tag{3-23}$$

$$\mathbb{E}[K(\boldsymbol{x}, \boldsymbol{x}_i)] = \int K(\boldsymbol{x}, \boldsymbol{x}_i) p(\boldsymbol{x}) d\boldsymbol{x} \tag{3-24}$$

$$\mathbb{E}[K(\boldsymbol{x}, \boldsymbol{x}_i) K(\boldsymbol{x}, \boldsymbol{x}_j)] = \int K(\boldsymbol{x}, \boldsymbol{x}_i) K(\boldsymbol{x}, \boldsymbol{x}_j) p(\boldsymbol{x}) d\boldsymbol{x} \tag{3-25}$$

The two ways of approximating the integrals are through a Taylor expansion or through exact moment matching.

b) Taylor approximations: By taking a Taylor approximation and ignoring all terms above first-order, the mean and variance functions of the Gaussian approximation in Equation 3-20 is given by [27]

$$m(\hat{\boldsymbol{x}}, \Sigma_x) = \mu(\hat{\boldsymbol{x}}) \tag{3-26}$$

$$v(\hat{\boldsymbol{x}}, \Sigma_x) = \Sigma_x(\hat{\boldsymbol{x}}) + \Delta\mu(\hat{\boldsymbol{x}}) \Sigma_x \Delta\mu(\hat{\boldsymbol{x}})^T \tag{3-27}$$

In comparison with the Naive approach from a), the mean function is the same but the variance has an added term giving an increase proportional to the variance of the input (which is a measure of how "off" the previous prediction can be) and to the gradient of the mean function (since input noise will cause more error for a steeper function rather than a flat function). Higher order Taylor approximations are also possible, at the cost of more computations.

c) Exact moment matching: For the case of the SE kernel, which is the most used in practice, exact evaluations of the integrals in Equations 3-23 - 3-25 have been derived [26]. The mean and covariance distributions are given by the following expressions

$$m(\hat{\boldsymbol{x}}, \Sigma_x) = \sum_{i=1}^{N} \beta_i c \mathcal{N}_{\hat{x}}(\boldsymbol{x_i}, \Lambda + \Sigma_x) \tag{3-28}$$

$$v(\hat{\boldsymbol{x}}, \Sigma_x) = K(\hat{\boldsymbol{x}}, \hat{\boldsymbol{x}}) - c^2 \sum_{i,j=1}^{N} (K_{ij}^{-1} - \beta_i \beta_j) \mathcal{N}_{x_i}(\boldsymbol{x}_j, 2\Lambda) \mathcal{N}_{\hat{x}}\left(\frac{\boldsymbol{x}_i + \boldsymbol{x}_j}{2}, \Sigma_x + \frac{\Lambda}{2}\right) - m(\hat{\boldsymbol{x}}, \Sigma_x)^2 \tag{3-29}$$

with $c = (2\pi)^{\frac{D}{2}} |\Lambda|^{\frac{1}{2}} C(\hat{\boldsymbol{x}}, \hat{\boldsymbol{x}})$.

They can be interpreted as optimally fitting a Gaussian distribution to the (unknown) distribution $p(\boldsymbol{y}|\hat{\boldsymbol{x}}, \Sigma_x)$.

A comparison between the different posterior distributions is shown in Figure 3-2. An example of propagating the variance through a multiple-step-ahead prediction is shown in Figure 3-3 for a double integrator system. A comparison between the predictive performance of all three methods is shown in Figure 3-4. In general, the naive approach should be used when only the predictive mean is of importance, since the corrective changes of this mean for the moment matching approach are not very impactful. If the variance is important, then moment matching is the most accurate method and a Taylor expansion is recommended if the SE kernel is not in use or if decreasing the computational time is desirable [28].

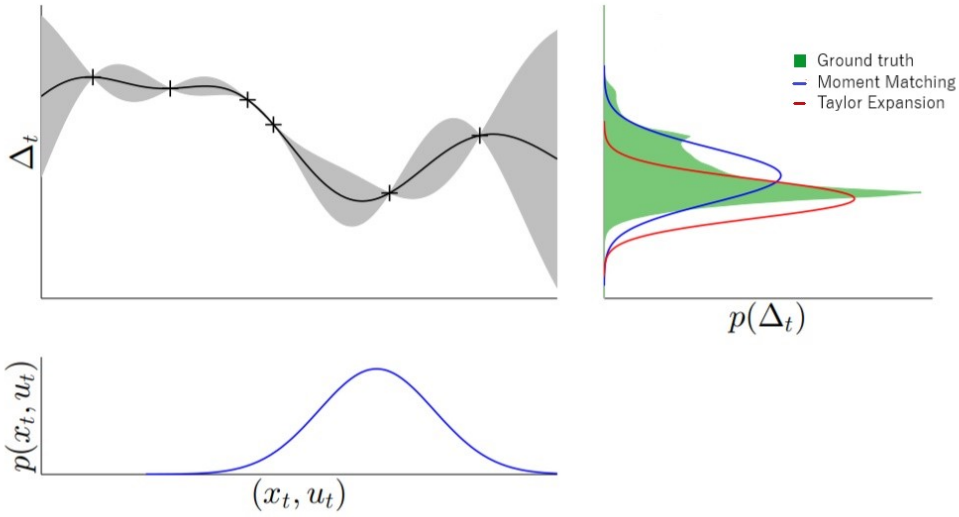

Figure 3-2: True distribution (estimated using MC methods) and approximate normal distributions given by Linearization and Moment Matching. [27]

These methods have been extended to be used in combination with some sparse approximations ([30], [31]), such as the Fully Independent Training Conditionals (FITC) approximation which will be considered in Chapter 4. Figure 3-5 shows that the adapted moment matching method for sparse approximations can give similar results in terms of predictive performance, while using ten times fewer points at prediction time.

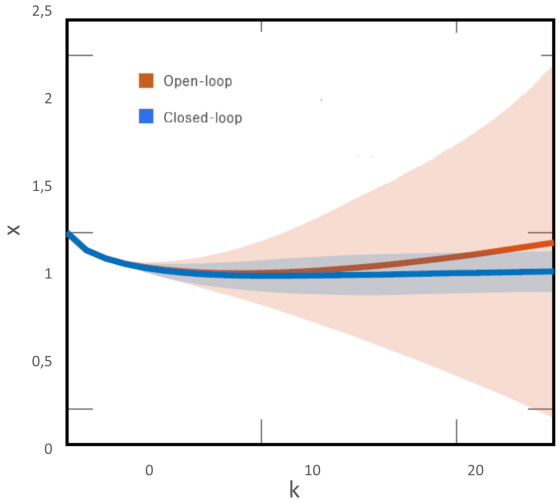

Figure 3-3: Propagation of variance for a double integrator system, in open-loop and closed-loop state feedback control. [13]

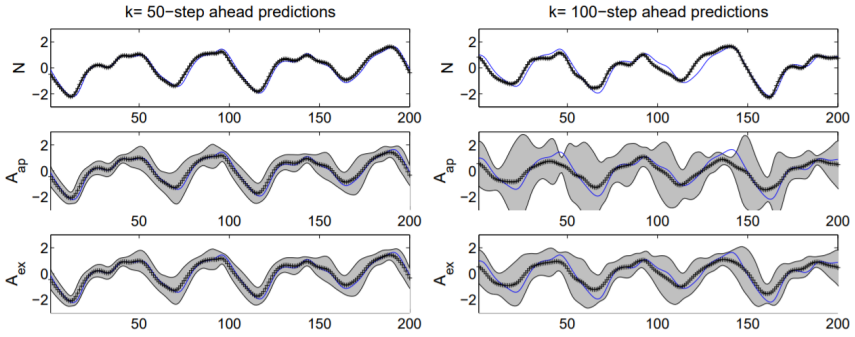

Figure 3-4: Comparison of the a), b) and c) methods for k-steps-ahead predictions; Left for k=50 and right for k=100. [29]

3-3 Dealing with noise in the training data

The previous Section discusses ways to make predictions from uncertain Gaussian distributed inputs. While this is useful at prediction time, it is only useful for making predictions and it assumes that the data used to train the GP is known and noise free. In System Identification and time series modelling, this is often not true since the output values in the data set are corrupted by some noise coming from the sensors. Variational inference methods can be used to train GPs with noisy inputs [25], but these methods tend to be slower to implement. This Section will focus on simpler and computationally faster methods, such as the Noisy Input Gaussian Process (NIGP) algorithm [32].

The NIGP method, introduced by [32], was especially designed for application with time series data where the measured outputs entering in the regressor vector are corrupted by some i.i.d. Gaussian distributed noise. The main idea of the algorithm is that the difficulty of dealing with noisy inputs is circumvented by shifting the effect of the uncertainty to the outputs.

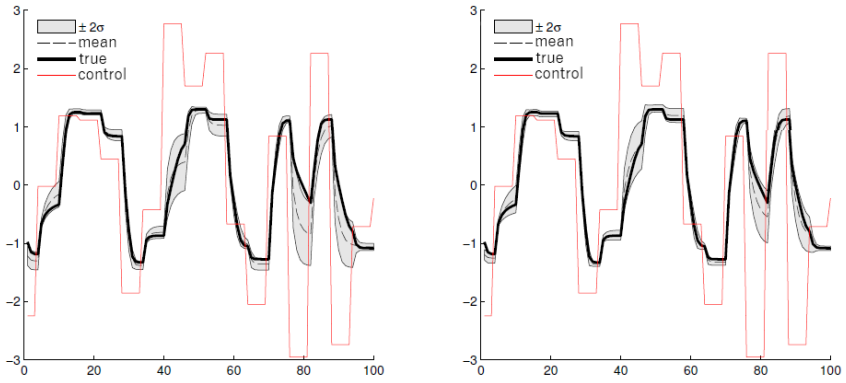

Figure 3-5: Comparison of the k-step-ahead performance using a full GP distribution with 200 training points and a FITC approximation with 20 inducing points chosen through joint optimization of the marginal likelihood (see Section 4-2-2). [30]

The intuition behind the algorithm is that if a function is steep with large derivative values, a small deviation in the input will result in a large deviation in the output. Conversely, a flat function shows similar output values for different neighbouring inputs.

Considering once again the problem of noisy input values

$$y = f(\boldsymbol{x}) + \epsilon_y \tag{3-30}$$
$$\boldsymbol{x} = \hat{\boldsymbol{x}} + \boldsymbol{\epsilon}_x \tag{3-31}$$

and expanding the function $f(\cdot)$ into a Taylor expansion and ignoring the higher level terms, we can write

$$f(\boldsymbol{x} - \boldsymbol{\epsilon}_x) \approx f(\boldsymbol{x}) + \boldsymbol{\epsilon}_x \boldsymbol{\partial}_{\bar{f}} \quad . \tag{3-32}$$

Following the notation from the paper, $\boldsymbol{\partial}_{\bar{f}}$ is a vector of derivatives of the mean function at $\boldsymbol{x}$ for each input and $\Delta_{\bar{f}}$ is an $N \times D$ matrix of such derivatives for all the points in the data set. The derivatives are taken w.r.t. the mean function value, which corresponds to ignoring the uncertainty on the derivatives. It follows that

$$y = f(\boldsymbol{x}) + \epsilon_y + \boldsymbol{\epsilon}_x \boldsymbol{\partial}_{\bar{f}} \quad . \tag{3-33}$$

This leads to a model where the noise term changes depending on the location of the input. Such a model is often called heteroscedastic. Heteroscedastic models are not uncommon in the GP literature, and they are useful in various applications, although this particular algorithm is aimed at dealing with time series data. The model simply corresponds to adding an additional noise term to the likelihood $p(y|f) = \mathcal{N}(f, \sigma_y^2 + \boldsymbol{\partial}_{\bar{f}}^T \Sigma_x \boldsymbol{\partial}_{\bar{f}})$, and then following the normal inference procedure. The posterior distribution will be

$$\boldsymbol{f}_* \sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_*) \tag{3-34}$$

$$\boldsymbol{\mu}_*(\boldsymbol{y}_*|X_*, X, \boldsymbol{y}) = K_{*f}\left(K_{ff} + \text{diag}\left([\sigma_y^2 I + \Delta_{\bar{f}}\Sigma_x\Delta_{\bar{f}}^T]\right)\right)^{-1}\boldsymbol{y} \tag{3-35}$$

$$\Sigma_*(\boldsymbol{y}_*|X_*, X, \boldsymbol{y}) = K_{**} - K_{*f}\left(K_{ff} + \text{diag}\left([\sigma_y^2 I + \Delta_{\bar{f}}\Sigma_x\Delta_{\bar{f}}^T]\right)\right)^{-1}K_{f*} \ . \tag{3-36}$$

The above expression is similar to the original GP but with a corrective term in the output scaling the input uncertainty with the function derivatives. Notice that to compute it requires access to the mean function derivatives. Therefore, the training requires first to get the full GP posterior, then to obtain the $\Delta_{\bar{f}}$ matrix, containing derivatives with respect to the inputs for each of the training points, and then to add the corrective term to the original GP posterior.

Access to the GP gradient of the mean GP function in 2-4 is given by

$$\frac{\partial \boldsymbol{\mu}_*}{\partial \boldsymbol{x}_*} = \frac{\partial}{\partial \boldsymbol{x}_*}\left[K_{*f}(K_{ff} + \sigma_n^2 I)^{-1}\right]\boldsymbol{y} \ . \tag{3-37}$$

The author of [33] provides an analytic expression of this gradient for the common case of an ARD SE covariance function (see Section 2-2-1):

$$\frac{\partial \boldsymbol{\mu}_*}{\partial \boldsymbol{x}_*} = -\Lambda^{-1}\tilde{X}_*^T\left[(K_{*f}(K_{ff} + \sigma_n^2 I)^{-1})^T \odot \boldsymbol{\alpha}\right] \tag{3-38}$$

where $\tilde{X}_* = \begin{bmatrix}\boldsymbol{x}_* - \boldsymbol{x}_1 & \dots & \boldsymbol{x}_* - \boldsymbol{x}_N\end{bmatrix}$, $\boldsymbol{\alpha} = (K_{ff} + \sigma_n^2 I)^{-1}\boldsymbol{y}$ and the operator $\odot$ represents an element-wise product.

The procedure in Equations 3-34 - 3-36 can even be iterated for more accurate results. Figure 3-6 shows the effects of the NIGP algorithm. An original full GP is learned in a) and then the derivatives are used to add the extra uncertainty coming from the noisy inputs. The final result is showed in b).

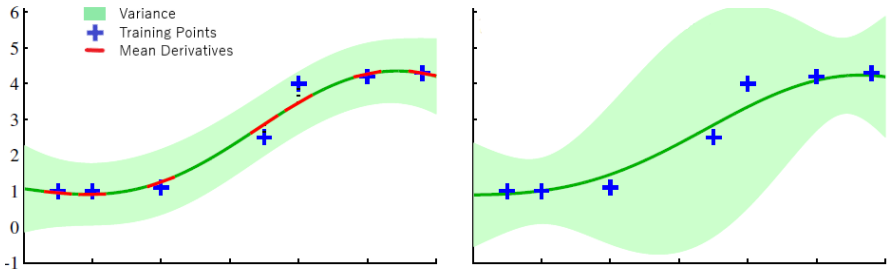

Figure 3-6: a) Full GP learned from considering noise-free inputs. b) Modified GP by adding a variance term accounting for the input uncertainty. [32]

After training and obtaining the posterior distribution, predictions can be made using the methods for propagation of variance introduced in the Section 3-2.

In [34] the NIGP method is adapted to be used in online applications, in conjunction with the FITC approximation (see Chapter 4).

3-4 Adaptations for online learning

All the framework discussed so far considers that the learning procedure happens offline, with previously measured data from the input/output response of the system. However, in certain applications it can be necessary for the GP to learn new dynamics online, either because the unknown system is time-varying or because the algorithm only models a limited region of the input space and one wants to expand the operating system to other unknown regions. Updating the learned model requires updating the data set, since a GP is a non-parametric model that stores the information in the data used for prediction. Updating the data set can be done by adding or removing a new data point from the set. Removing data points is necessary if the data set starts becoming prohibitively large. The two questions that arise are how to chose which data points to add/remove and how to efficiently implement data points. These issues have been discussed e.g. in [35], [34] and [36].

Since the GP suffers from scaling problems, an increased data set might not be feasible for online implementations. For that reason, it's important to consider the following problems: when to update the GP prediction by including a data point, or by removing a data point; and how to efficiently perform an update.

One way to deal with the constant stream of new data is to consider only the most recent data received and forget about older data, which is a technique known as *windowing*. This will lead the GP to constantly learn and forget certain regions of the input space depending on the operating point, or in the case of a time-varying system it will help erase data that does not accurately describe the new dynamics. Another option is to use one of the greedy criteria from Section 4-2-1 on the new data points, and add them to the prediction if they contribute with new valuable information. This method has the advantage that not all the data points coming in will be integrated, which means that fewer updates have to be done. For example, in [36] it's proposed to add a new point $\{\boldsymbol{x}_{i+1}, y_{i+1}\}$ if the prediction error $e_{i+1} = y_{i+1} - \mu_{i+1}$ exceeds some predetermined maximum value e_{max}, and if the data set exceeds a threshold number of points ($|D| > N_{max}$) the oldest point in the data set is removed. Figure 3-7 shows an empirical example of how the computations for a one-step-ahead prediction for a full GP can vary with the number of data points.

After the decision to include a new data point, the distribution of the predictive GP must be updated. The author of [36] provides an efficient way to perform updates, which is introduced next.
To compute the predictive posterior distribution of the GP, Equations 2-9 and 2-10 for the posterior mean and covariance functions are repeated here for convenience

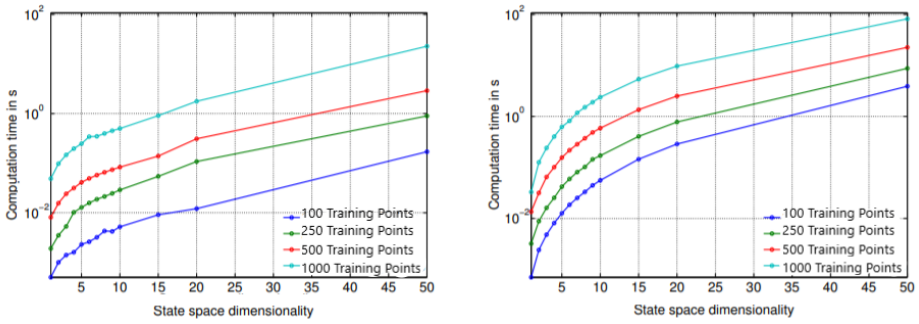

Figure 3-7: Empirical change in computational time for a one-step-ahead prediction as a function of the number of data points for a) A Linearized approximation b) Exact moment matching. [27]

$$p(\boldsymbol{f_*}|X_*, X, \boldsymbol{y}) = \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_*) \tag{3-39}$$

$$\boldsymbol{\mu_*} = \boldsymbol{m}_* + K_{*f}K_{ff}^{-1}(\mathbf{y} - \boldsymbol{m}_*) \tag{3-40}$$

$$\Sigma_* = K_{**} - K_{*f}K_{ff}^{-1}K_{f*} \quad . \tag{3-41}$$

The computationally efficient way of solving these Equations is by using the Cholesky decomposition. The Cholesky decomposition of a positive definite matrix A is $A = R^T R$, where $R = \text{chol}(A)$ is upper triangular. This allows to solve the algebraic Equation $x = A^{-1}b : A\backslash b$ as $x = R\backslash(R^T\backslash b)$. To compute the predictive distribution we define

$$\boldsymbol{\alpha} := K_{ff}^{-1}(\mathbf{y} - \boldsymbol{m}_*) \tag{3-42}$$

$$\beta := K_{ff}^{-1}K_{f*} \tag{3-43}$$

and considering $R = \text{chol}(K)$ these two terms can be computed as

$$\boldsymbol{\alpha} := R\backslash(R^T\backslash(\mathbf{y} - \boldsymbol{m}_*)) \tag{3-44}$$

$$\beta := R\backslash(R^T\backslash K_{*f}) \quad . \tag{3-45}$$

The computation of R can be done in $\mathcal{O}(N^3)$ while $\boldsymbol{\alpha}$ and β can be computed in $\mathcal{O}(N^2)$. In offline methods R and $\boldsymbol{\alpha}$ only need to be computed once, while β needs to be computed for every test point. In online methods, There is no need to recompute the Cholesky decomposition. Instead, an efficient update [36] can be obtained by considering the kernel matrix in a block diagonal form *before* adding new point(s)

$$K_{ff} = \begin{bmatrix} K_{11} & K_{13} \\ K_{13}^T & K_{33} \end{bmatrix} \tag{3-46}$$

$$R_{ff} = \begin{bmatrix} R_{11} & R_{13} \\ 0 & R_{33} \end{bmatrix} \tag{3-47}$$

and *after* adding new point(s)

$$K_{ff} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{12}^T & K_{22} & K_{23} \\ K_{13}^T & K_{23}^T & K_{33} \end{bmatrix} \tag{3-48}$$

$$S_{ff} = \begin{bmatrix} S_{11} & S_{12} & S_{13} \\ 0 & S_{22} & S_{23} \\ 0 & 0 & S_{33} \end{bmatrix} . \tag{3-49}$$

The updated Cholesky matrix S_{ff} can be computed using

$$S_{11} = R_{11} \qquad S_{22} = \text{chol}(K_{22} - S_{12}^T S_{12}) \qquad S_{33} = \text{chol}(R_{33}^T R_{33} - S_{23}^T S_{23})$$

$$S_{12} = R_{11}^T \backslash K_{12} \qquad S_{13} = R_{13} \qquad S_{23} = S_{22}^T \backslash (K_{23} - S_{12}^T S_{13})$$

This allows to efficiently update the Cholesky factorization matrix R in $\mathcal{O}(1)$ time, while the computations of $\boldsymbol{\alpha}$ and β remain at $\mathcal{O}(N^2)$. However, this algorithm implies that the hyper-parameters are determined offline, since different hyper-parameter values would change every entry of the kernel matrix. In fact, due to the large computational effort there seems to be no research so far that performs updates of the GP hyper-parameters in real time. Regardless, this framework allows to efficiently incorporate new data points in the GP posterior prediction.

It's important to emphasize the difference between what can and cannot be achieved in an online fashion through the above mentioned methods. A new point can be added to the training data kept in storage throughout the experiment, however kernel hyper-parameters are always determined offline[1]. This means that exploration of untrained regions can be done online only if said regions of the input space have a behaviour that can be accurately represented through the pre-determined hyper-parameters. For example, if a function is slow-varying in the offline training region and the optimized length-scales are large, moving to a region of the input space where the function becomes quick-varying and rough will result in poor model predictions even if training points are later added to the training data set.

[1] If the reader is interested in an algorithm that can on some level deal with hyper-parameter updates in real time, the author of [37] has proposed an algorithm that can incorporate these updates through Variational Inference methods.

Chapter 4

Gaussian processes for big data

The basic method for Gaussian Process Regression (GPR) presented in Chapter 2 is already a powerful tool for obtaining posterior function distributions by combining prior beliefs about the function and noisy observations of the function output, and preforming inference based on a Bayesian framework. GPR can and has been applied to a wide variety of fields in the literature. One of the biggest challenges with the method is that computations of the inverse kernel matrix K_{ff}^{-1} (in Equations 2-9 and 2-10) scale in the order of $\mathcal{O}(N^3)$ with the number of training points. After that, predictions at a test point scale with $\mathcal{O}(N)$ for the mean value and with $\mathcal{O}(N^2)$ for the variance value. This imposes restrictions on the applicability of Gaussian Process (GP)s when the number of test points reaches the order of thousands.

Solutions for implementing GPs in a computationally cheaper way have been the target of much research in the last two decades. One way to cope with it is simply by taking advantage of the increasing computational resources that arise from better computer processing. Nevertheless, alternative solutions to apply GPR to larger data sets are necessary, and some of them are presented in this Chapter.

The author of [38] recently provided an overview of the most popular techniques for dealing with this problem of dimensionality. Figure 4-1 shows the percentages of each method in the current literature.

The choice of which method to use in a given practical application is not obvious and it should in general depend on practical requirements of the user, such as computational time, quality of prediction, choice of kernel function, training data, and the overall behaviour of the underlying function. While much analysis can be done over each of these methods, this thesis will focus only on global approximation methods, specifically prior approximations and sparse kernel methods. This is because these methods have been used the most on previous research involving control applications (see e.g. [28] for a prior method example and [15] for a sparse spectrum example).

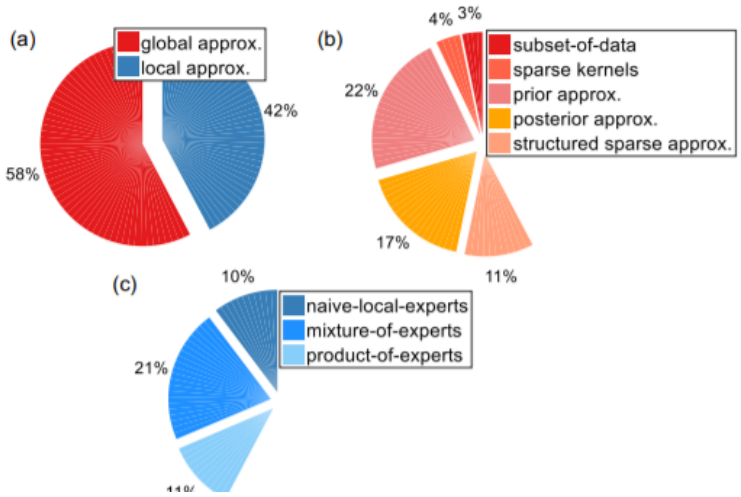

Figure 4-1: Overview of the current research on GP approximations in the literature. [38]

To briefly mention other methods, local approximation methods are particularly useful in the estimation of a system where the underlying dynamics change drastically across different regions of the input space, or when the user wants to divide the modelling into subsections containing (smaller) local models. Structured sparse approximations, which explore the nature of kernels with a specific structure, often require the training data to be collected on a well-defined grid in the input space, which is typically not a realistic possibility in System Identification. Posterior approximations are an interesting field of research which makes use of methods such as Markov Chain Monte Carlo (MCMC) Sampling [24] and Variational Inference [25], and have had a lot of attention in recent years. However, they can be slow to implement in an online fashion and have yet to see much use in such applications.

This Chapter will introduce sparse approximations as a way to deal with the computational costs of increasingly large data sets. Two methods will be introduced to understand the way to achieve this computational reduction: an *inducing points* method and a *sparse kernel* method. Section 4-2 will introduce the Fully Independent Training Conditionals (FITC) approximation, which is regarded as the most popular inducing points method for prior approximations (see Figure 4-1), and Section 4-3 will introduce the Sparse Spectrum (SS) approximation, belonging to the category of sparse kernel methods. Section 4-4 will compare the two methods in terms of strengths and performance, which will be helpful later on to make a decision of which approximation method to use in tests and practical implementations.

4-1 Inducing Points methods

GP approximations through inducing points are a popular way of dealing with the problem of scalability. The idea is to take advantage of a subset of regressors $M << N$ to make statistical assumptions (approximations) on the probability distributions that allow to perform some type of dimensionality reduction, where predictions can be done through the inducing

inputs rather than through the entire set of N data points. This Section will quickly introduce the mathematical background behind these approaches, as well as the FITC algorithm. Other algorithms that fit this same background are the Subset of Regressors (SoR) [39], the Deterministic Training Conditionals (DTC) [40] and the Partially Independent Training Conditionals (PITC) [41]. Although these methods were originally derived in different ways, they were later rewritten in the same unifying framework by the author of [41]. It was shown that they can all be derived naturally as approximations on the GP prior followed by the exact inference and posterior conditioning.

In general, since inference comes from combining a prior with a likelihood, approximations can be shifted from one or the other depending on the perspective of the author. In this Section the method will be presented as approximations of the prior, since this allows to explain similar methods through a unifying framework.

4-1-1 Subset of Data

Let us begin with the simplest idea to approximate the full GP. This consists of picking a subset of data points $\mathbb{I}_M \subset \mathbb{I}_N$ of size $M << N$ and doing normal inference using this subset of data. Considering $\mathbf{u}$ to be the targets corresponding to inputs $\boldsymbol{x}_M$, then

$$\boldsymbol{\mu}_* = K_{*u}(K_{uu} + \sigma_n^2 I)^{-1}\boldsymbol{u} \tag{4-1}$$

$$\Sigma_* = K_{**} - K_{*u}(K_{uu} + \sigma_n^2 I)^{-1}K_{u*} \ . \tag{4-2}$$

Of course the use of this method is somewhat counter-intuitive. It is essentially dealing with the problem of having a large data set by choosing to work with only a small subset of data points, which means that the excluded data points add no information to learning the function. Ultimately, having available data is a benefit and trying to incorporate the entire data set in the learning process is important. However, this method brings the computational burden to $\mathcal{O}(M^3)$, which is a more significant reduction than any other method considered in the remaining of this Chapter. One could use the rest of the computational "budget" to solve for the choice of location for the M points, such that the approximation is optimized. For example, by picking points that are close in distance to the desired point prediction, and disregarding points that are far from it (which is reminiscent of the concept of support vectors in the SVM framework, where sparse solutions exist because only a few data points have non-zero contributions to the solution).

Therefore, a smart choice of the subset of M points could potentially justify the use of this simple method (see [42]). Nevertheless, Section 4-1-2 will introduce a way to maintain the information on the remaining set of $N \backslash M$ data points.

4-1-2 Fully Independent Training Conditionals

This method can be considered as a GP learned after performing approximations on the prior $p(\boldsymbol{f}, \boldsymbol{f}_*)$, and then conducting inference in the normal way. First of all, the method assumes

that the joint distribution between the training data and the test data is independent given $\boldsymbol{u}$:

$$p(\boldsymbol{f}, \boldsymbol{f}_*) = \int p(\boldsymbol{f}, \boldsymbol{f}_*, \boldsymbol{u}) d\boldsymbol{u} = \int p(\boldsymbol{f}, \boldsymbol{f}_* | \boldsymbol{u}) p(\boldsymbol{u}) d\boldsymbol{u} \tag{4-3}$$

$$\approx \int \tilde{p}(\boldsymbol{f}|\boldsymbol{u}) \tilde{\boldsymbol{p}}(\boldsymbol{f}_*|\boldsymbol{u}) p(\boldsymbol{u}) d\boldsymbol{u} \tag{4-4}$$

where the first equality comes from the consistency property of a GP, which allows to recover the original distribution $p(\boldsymbol{f}, \boldsymbol{f}_*)$ by taking $p(\boldsymbol{f}, \boldsymbol{f}_*, \boldsymbol{u})$ and marginalizing out $\boldsymbol{u}$. The above result shows the general meaning of the inducing variables in the framework of approximating priors: they establish the statistical dependencies between the training points and the test points. This concept is represented graphically in Figure 4-2, where the lines between variables represent statistical dependencies. There is no line connecting the points $\boldsymbol{f}$ with the point(s) $\boldsymbol{f_*}$, showing that they are only related through the inducing variables $\boldsymbol{u}$.

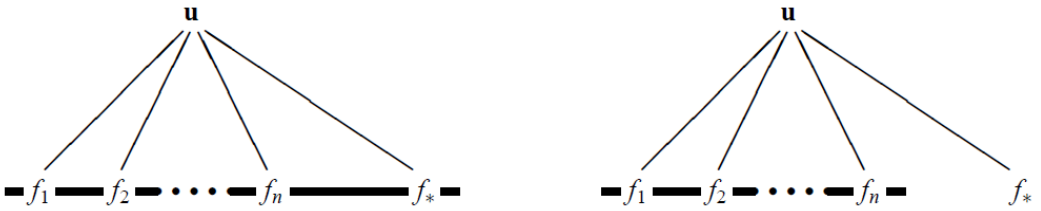

Figure 4-2: Graphical representation of the statistical dependencies between training and test points for sparse methods. On the left is the full GP; On the right is the approximation that targets $\boldsymbol{f}$ and $\boldsymbol{f_*}$ only communicate through $\boldsymbol{u}$. [41]

The above assumption is taken for all inducing points methods, and then further approximations are taken for each individual method. For the FITC method, originally introduced by the author of [43], the approximation consists in the fact that every training point is independent from each other when conditioned on the inducing variables

$$\tilde{p}(\boldsymbol{f}|\boldsymbol{u}) = \prod_{i=1}^{N} p(f_i|\boldsymbol{u}) = N\left(K_{fu}K_u^{-1}\boldsymbol{u}, \operatorname{diag}\left([K_{ff} - Q_{ff}]\right)\right) \tag{4-5}$$

$$\tilde{p}(\boldsymbol{f_*}|\boldsymbol{u}) = p(\boldsymbol{f_*}|\boldsymbol{u}) \ . \tag{4-6}$$

Remembering that the prior on the inducing targets $\boldsymbol{u}$ is $p(\boldsymbol{u}) \sim N(\boldsymbol{0}, K_{uu})$, the integral in Equation 4-4 solved to compute the approximate prior

$$\tilde{p}_{\text{FITC}}(\boldsymbol{f}, \boldsymbol{f_*}) = \begin{bmatrix} Q_{ff} - \operatorname{diag}[Q_{ff} - K_{ff}]) & Q_{f*} \\ Q_{*f} & K_{**} \end{bmatrix} \tag{4-7}$$

using the common notation of $Q_{ab} = k_{au}K_{uu}^{-1}k_{ub}$, which is often called the Nyström approximation [44]. After obtaining the approximate prior, inference is carried out at usual. The approximate posterior is

$$\tilde{p}_{\text{FITC}}(\boldsymbol{f_*}|\boldsymbol{y}) = N(\boldsymbol{\mu_*}, \Sigma_*) \tag{4-8}$$

$$\boldsymbol{\mu_*} = Q_{*u}(Q_{ff} + \Lambda)^{-1}\boldsymbol{y} \tag{4-9}$$

$$\Sigma_* = K_{**} - Q_{*u}(Q_{ff} + \Lambda)^{-1}Q_{f*} \tag{4-10}$$

Using the notation that $\Lambda = \text{diag}[K_{ff} - Q_{ff} + \sigma_n^2 I]$.

Now the computations are of the order of $\mathcal{O}(NM^2)$, since its only required to invert the $M \times M$ matrix K_{uu} that appears on the Nyström approximation. Therefore, the FITC method achieves a computationally cheaper way to make predictions while still maintaining information on all the points in te data set. The computational simplifications are mainly focused at the inversion of the K_{ff} matrix, which is the term that accounts for most of the complications regarding scalability. Despite the statistical approximations, this method tends to outperform the SoR and DTC methods because it still retains the diagonal elements from the full GP kernel for the training data.

4-2 Choosing the location of inducing points

The previous inducing points method attempts to soften computations by taking advantage of statistical dependencies between the data set and a relatively low number of inducing points $\boldsymbol{x}_M$. Until now, nothing has been said about the location of these inducing points. The choice of inducing points is however critical to the success of the method. This Section will explore ways that have been used to determine these inducing point locations.

It's important to realise that in the general case the points $\boldsymbol{x}_M$ do not have to be a subset of the training points (since the posterior distributions are never dependent on $\boldsymbol{u}$ itself). This idea was only introduced by [43] as part of the Sparse Pseudo-input Gaussian Process (SPGP) algorithm, where it was proposed to optimize the location of the inducing points by letting them change continuously in the input space. The logic behind it is that optimizing a discrete set of points in a large data set is a prohibitively big combinatorial optimization problem. By allowing the point locations to vary continuously, the optimization problem is relaxed and therefore easier to solve.

In general there are three major ways to select the location of inducing inputs. The easier one is to chose them randomly or by some clustering criterion, i.e. chose a set of inputs form the training set that are well spread out over the input space. The second one is to pick some sort of greedy criterion to measure the benefit of adding a new point i to the existing set of input points, and adding points that seem to be informative based on that criterion. The third method is to optimize the location of the inducing points along with the kernel

hyper-parameters. The next sections elaborate on the second and third methods.

In some applications, such as Predictive Control, it is also possible to use the so-called transductive selection of inducing inputs, where the input locations are chosen to coincide with the test points. This concept is applied in [28] for Model Predictive Control (MPC), where the inducing points are simply taken as the reference points for the trajectory.

4-2-1 Greedy selection of inducing points

These methods select inducing points from the available training points $\mathbb{I}_N$. Numerous greedy methods have been introduced in the literature, such as matching pursuit [45], informative gain [46], sparse spectral sampling, etc. A greedy selection criterion sequentially introduces a new inducing point into a data set $\mathbb{I}_M$ (up to a maximum of $M \ll N$) by looking through the training data points $\mathbb{I}_{N \setminus M}$ and computing some measure Δ_i, $i \in \mathbb{I}_{N \setminus M}$, for how informative the point will be. The process of greedy selection is as follows:

- Begin with $\mathbb{I}_M = \emptyset$
- Compute Δ_i for some (possibly all) points $i \in \mathbb{I}_{N \setminus M}$.
- Pick $\Delta_{i^*} = \text{argmax}\Delta_i$ and add it to the active set: $I \leftarrow I \cup \{i^*\}$
- Repeat until $|\mathbb{I}_M| = M$

Several choices of measuring greedy criteria have been proposed. For example, [47] proposed to use the change in entropy as a measure

$$\Delta_i = H\left[\tilde{p}_{\mathbb{I} \cup \{i\}}\right] - H\left[\tilde{p}_{\mathbb{I}}\right] \tag{4-11}$$

Where $H\left[\tilde{p}_{\mathbb{I}}\right] = \frac{1}{2} log\left|2\pi Q_{ff}\right|$ is defined as the entropy, and adding an extra inducing point will cause its value to increase.
The author of [48] considers a different measure based on expectation maximization, with the criteria

$$\Delta_i = |y_i - \tilde{\mu}_i| \tag{4-12}$$

Where $\tilde{\mu}_i$ is the mean estimate of the sparse approximation. This criterion focuses on the predictive capabilities of the sparse GP on the location of the training points and adds the point that is giving the highest predictive error.
Another option [45] is to use the approximated negative log marginal likelihood of Equation 2-21, given by

$$\tilde{F}(\boldsymbol{\theta}, \mathbb{I}_M) = \frac{1}{2}\boldsymbol{y}^T(Q_{ff} + \sigma_n^2 I)^{-1}\boldsymbol{y} + \frac{1}{2}\log\left|Q_{ff} + \sigma_n^2 I\right| \tag{4-13}$$

and use the measure

$$\Delta_i = \left| F(\theta; \mathbb{I}_M) - F(\mathbb{I}_{M\cup\{i\}}) \right| \tag{4-14}$$

Similar to the process of learning hyper-parameters, here the marginal likelihood will be used as a measure to learn the quality of the inducing points. One disadvantage of any greedy criterion is that the inducing points are added to the set assuming that the hyper-parameters for the full GP were already learned and fixed. Alternatively, iterating between adding points to the active set and re-learning hyper-parameters can be a good option. Either of these approaches carries the risk of over-fitting, since the number of effective hyper-parameters is augmented and the information criteria is trying to minimize the log likelihood, which is influenced by the training data.

Different methods may have different approaches to adding and deleting points from the inducing set, but greedy selection methods have in common that they use a criteria (in this case the negative log likelihood) to access the value of points and decide whether to include them in the active set. While many criteria exists, a proper comparison of different selection criteria is yet to be done.

4-2-2 Optimization methods

As an alternative to greedy selection, [43] proposes an alternative way of finding inducing point locations by optimizing these locations along with the kernel hyper-parameters. For this method, the inducing points $\boldsymbol{x}_M$ no longer have to be a subset of training points. Instead, we allow them to vary continuously in the optimization problem, which is much simpler than performing discrete optimization. Of course this would only work if considering a continuous input space makes sense in the context of the problem.
Under this perspective, the inducing inputs are simply extra variables to be optimized along with the hyper-parameters. Therefore, in the process of determining the hyper-parameter optimal values $\hat{\boldsymbol{\theta}}$, we should also add extra M optimization variables for the input locations and use the approximate log marginal likelihood

$$\log q(\boldsymbol{y}|X_M) = -\frac{1}{2}\boldsymbol{y}^T(Q_{ff}+\Lambda)^{-1}\boldsymbol{y} - \frac{1}{2}\log|Q_{ff}+\Lambda| - \frac{n}{2}\log(2\pi) \tag{4-15}$$

Where $\Lambda = \text{diag}[K_{ff} - Q_{ff}] + \sigma_n^2 I$ for the FITC method. Since this optimization adds M extra effective hyper-parameters to the problem it can be prone to over-fitting and it can be subject to problems with local minima resulting from an optimization problem of very high dimensionality. In practice, the method is useful if the GP is very sparse and there is a low amount of inducing inputs M, and it loses its effectiveness for increased values of M.

4-3 Sparse kernel methods

The second major class of algorithms considered are sparse kernel approximations. This was first introduced in [49]. This paper will follow the work of [50], which introduces the Sparse

Spectrum (SS) approximation.

Unlike Section 4-1 where the computational time save was achieved through the assumption of statistical independencies between most points in the data set and using inducing points to bridge the information between the training data set and the test predictions, sparse kernel methods reduce the computational burden by approximating the kernel function itself. Specifically it makes use of a consequence of the Weiner-Khinchin theorem, in which the covariance function can be equivalently represented as a power spectrum $S(s)$, or in other words the covariance function and the power spectrum are a Fourier-transform pair. The main condition for this is that the covariance function has to be stationary, such that $K(x, y) = K(x - y) = K(\tau)$ for $\tau = x - y$. In this case we can write

$$K(\tau) = \int S(s) \exp(2\pi i \tau s) ds \tag{4-16}$$

$$S(s) = \int K(\tau) \exp(-2\pi i \tau s) d\tau \quad . \tag{4-17}$$

It is therefore equivalent to approximate the power spectrum of the covariance function. The limitation imposed by the fact that the covariance function should be stationary is not a serious limitation for the work of this paper since the covariance function used throughout is the Automatic Relevance Determination (ARD) Squared Exponential (SE) function introduced in 2-2-1. In general the most used covariance functions have this stationary property, such as the Matérn class and the Periodic kernel.

The power spectrum $S(s)$ can be shown to be proportional to some probability density function $p(s)$, which allows to rewrite the covariance function between two points as

$$K(\tau) = \sigma_0^2 \mathbb{E}_{p_s} \left[e^{2\pi i s^T x_i} (e^{2\pi i s^T x_j})^* \right] \quad . \tag{4-18}$$

Equation 4-18 is just another way of representing a stationary covariane function, this time as a function of the power frequencies. The authors of [50] approximate the expected value over p_s by randomly sampling some frequencies from the probability distribution p_s and using the Monte-Carlo method to obtain an approximate expected value in Equation 4-18. If furthermore the sampled frequencies are symmetric with respect to zero, meaning that the sampling is done in pairs $\{s_r, -s_r\}$, then the complex term will cancel out and the covariance function function can be approximated by

$$K(\tau) = \frac{\sigma_0^2}{m} \sum_{r=1}^{m} \cos(2\pi s_r^T \tau) \tag{4-19}$$

where m is the number of sampled frequencies. This set of sampled frequencies is often called spectral points. Figure 4-3 shows a visual comparison between the SE kernel and two approximations.

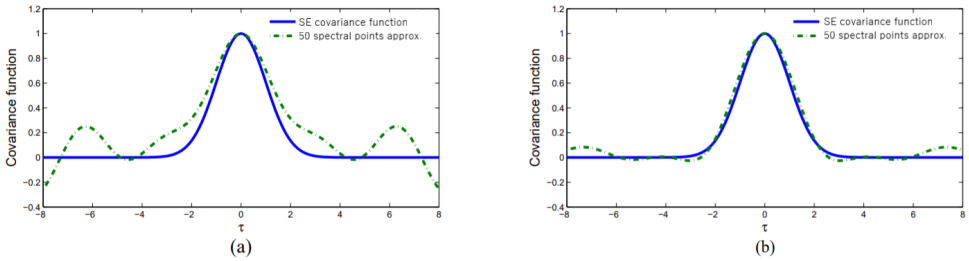

Figure 4-3: SE covariance function and its approximations for a) 10 sampled spectral points b) 50 sampled spectral points. [50]

The next section will focus on the method for obtaining these spectral points. After obtaining the spectral points, the author of 4-3 provides the equations for approximate test predictions:

$$p(\boldsymbol{f_*}|X_*, X, \boldsymbol{y}) = N(\boldsymbol{\mu_*}, \Sigma_*) \tag{4-20}$$

$$\boldsymbol{\mu_*} = \phi(X_*)^T A^{-1} \phi(X)^T \boldsymbol{y} \tag{4-21}$$

$$\Sigma_* = \sigma_n^2 \phi(X_*)^T A^{-1} \phi(X_*) \tag{4-22}$$

where $\phi(x) = \begin{bmatrix}\cos(2\pi s_1^T x) & \sin(2\pi s_1^T x) & \dots & \cos(2\pi s_m^T x) & \sin(2\pi s_m^T x)\end{bmatrix}$ is a feature map of length $2m$ and $A = \phi(X)^T\phi(X) + \sigma_n^2 I$. It is also possible to derive an approximation to the marginal log likelihood, which will now depend on the sampled frequencies in addition to the original kernel hyper-parameters:

$$L = \frac{1}{2}\log|A| - \frac{D}{2}\log\sigma_n^2 + \frac{n}{2}\log(2\pi\sigma_n^2) + \frac{1}{2\sigma_n^2}(y^T y - y^T\phi(X_*)A^{-1}\phi(X_*)^T y) \ . \tag{4-23}$$

Equations 4-20 - 4-22 are analogous to Equations 2-4 - 2-6 and equation 4-23 to Equation 2-21. Since the matrix A is now of size $2m \times 2m$, this requires the inversion of a smaller matrix if $2m < N$, and importantly the size of matrix A is no longer dependent on the number of training points.

4-3-1 Practical application

In practice the algorithm is implemented by sampling from the aforementioned probability distribution p_s. This probability distribution, for the case of the ARD SE kernel function, takes the form of:

$$p_s^{\text{ARD}}(s) = \sqrt{|2\pi\Lambda|}\exp\left(-2\pi^2 s^T\Lambda s\right) = \mathcal{N}(0, \Lambda^{-1}) \ . \tag{4-24}$$

Notice that the probability density depends on the length-scales Λ. This means that the length-scales have to be fixed in order to obtain the spectral points. The author of [50] recommends to jointly optimize the spectral points and the hyper-parameters by using equation 4-23. While this seems like a natural idea, it can lead to over-fitting and to exceedingly small prediction variances, which leads to overconfident predictions.

To avoid over-fitting, in this paper the kernel hyper-parameters are first optimized for the full GP, as described in Section 2-3. Then, the spectral points are sampled from the distribution in Equation 4-24. To ensure quality in the predictions, multiple batches of spectral points are sampled, and the best batch is chosen by picking the one that minimizes the log marginal likelihood equation 4-23. This will in part predict over-fitting, since the kernel hyper-parameters are fixed beforehand, and the approximation will tend to look more similar to the full GP.

4-4 Comparison between methods

This Section performs an empirical comparison between the two sparse methods introduced in this Chapter[1]: an inducing points method and a sparse kernel method[2]. This will hopefully help make the decision of which method to use later on in Chapter 6 for the test cases and practical implementations.

This comparison will be done based on a single experiment of the following 2-dimensional function:

$$y(\boldsymbol{x}) = x_1 \exp(-x_1^2 - x_2^2) \tag{4-25}$$

The function in Equation 4-25 is a simple yet effective benchmark in the literature for testing regression algorithms [53]. It's behaviour is highly nonlinear near the origin and it becomes flat when moving away from it (see Figure 4-4). The goal is to compare the predictive performance of three GP algorithms: the full GP; the FITC approximation; and the SS approximation.

For the experiment, points were collected in a 20×20 grid covering the region of $(x_1, x_2) \in [-2, 4]^2$. The FITC approximation uses M inducing points equally spaced throughout the input space[3]. The SS approximation uses M spectral frequencies sampled from the power spectrum in Equation 4-24. This will ensure that both methods have relatively similar computational complexities.

The comparison is based on two of the metrics introduced in Section 2-4: the Mean Squared Error (MSE) and the Mean Standardized Log Loss (MSLL). Table 4-1 shows the performance

[1] This thesis makes use of the GPML toolbox (see [51]).

[2] To a more in-depth theoretical comparison the reader is referred to [52].

[3] The FITC approximation could outperform if the inducing point locations were jointly optimized, as described in Section 4-2-2. However, this is not done here to avoid over-fitting, and also because later on in online applications it would become an extra burden to optimize inducing point locations for every prediction step.

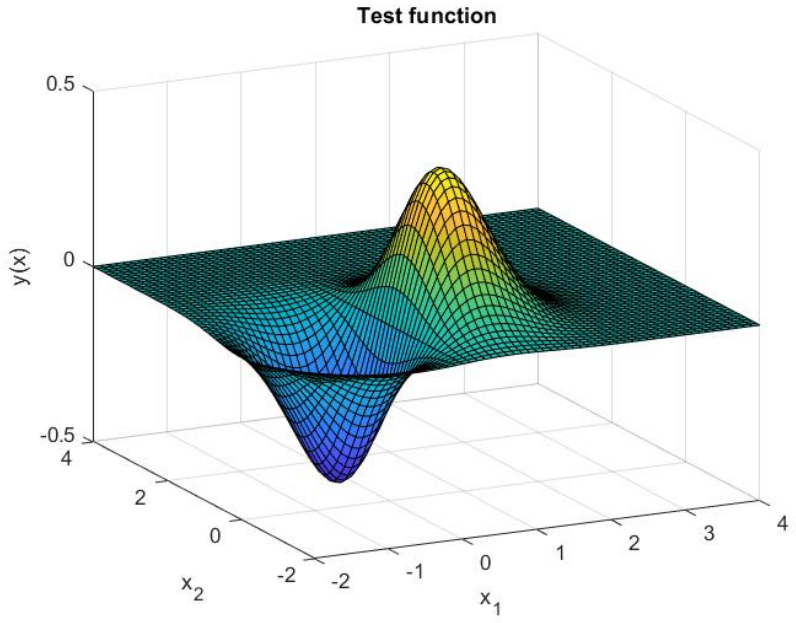

Figure 4-4: Graphical representation of the function in Equation 4-25 for $(x_1, x_2) \in [-2, 4]^2$.

	MSE			MSLL		
nº inducing points/ nº spectral frequencies	full GP	SS GP	FITC	full GP	SS GP	FITC
25	0.0001	0.0009	0.003	-1.75	-1.31	-0.75
36	0.0001	0.0003	0.0005	-1.5	-1.4	-1.1
49	0.0001	0.0003	0.0005	-1.6	-1.5	-1.3
64	0.0001	0.0002	0.0001	-1.7	-1.7	-1.6

Table 4-1: Comparison of predictive metrics for the sparse approximation algorithms, as a function of the number of inducing points (for the FITC approximation) and spectral frequencies (for the SS approximation).

of each algorithm for different computational complexities. All models are relatively unbiased in the validation phase, with a bias in the order of $\mathcal{O}(10^{-5})$. The SS algorithm seems to slightly outperform the FITC algorithm for lower numbers of M, both in the MSE and MSLL metrics, while the FITC algorithm outperforms for larger numbers of M. However, the SS approximation should theoretically outperform for examples using higher input dimension, since it becomes more difficult to pick the locations of inducing points and choosing them on a grid eventually becomes infeasible.

To face this problem of infeasibility, let us consider another experiment where the FITC inducing points are chosen at random as a subset of the training data set. In this setting, both the SS and the FITC approximations have an associated stochastic component. In the SS approximation, the sampled frequencies are sampled from a probability distribution (see Section 4-3-1) and in the FITC approximation the subset of inducing points is chosen at random. While the methods can be compared interms of performance, they are very different

in terms of the approach used for sparse approximation. The SS method approximates the *kernel function* through a distribution that is *independent* of the training set, while the FITC method approximates a *kernel matrix* through a selection process that is *dependent* on the training set. Understanding of the different approaches shows that a direct empirical comparison between the two methods is not trivial.

To illustrate this last point, consider a scenario where the predictive error is compared for both methods using the Root Mean Squared Error (RMSE) metric. To test for robustness of these stochastic predictors, the validation step was repeated 50 times, and the mean RMSE values are presented in a graph along its standard deviation. The validation results of the function in Figure 4-4 are shown in Figure 4-5. These results re-state that the SS approximation outperforms for lower values of sampled frequencies/inducing points, while the FITC approximation outperforms for higher values. Not only that, but the standard deviation is smaller, which on the one hand shows that the algorithm is more robust to stochastic variations. On the other hand, it shows that "good" choices of inducing points can severely increase the predictive accuracy of the FITC algorithm.

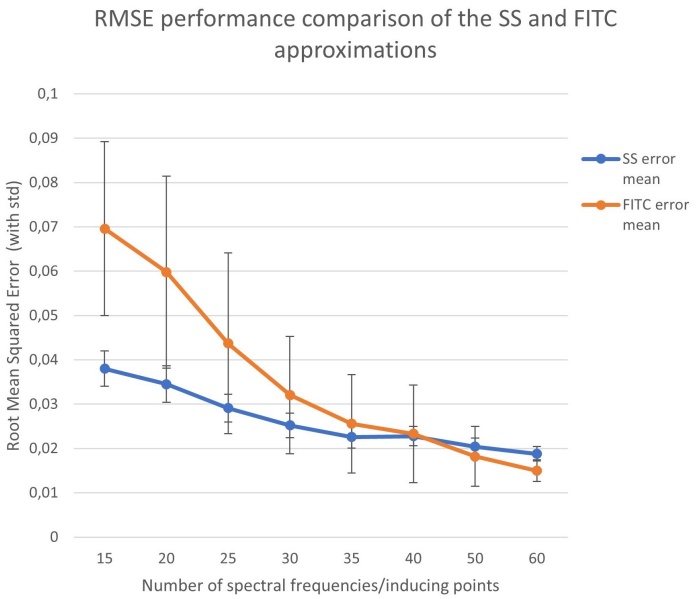

Figure 4-5: Validation results for a comparison of the FITC and SS approximations for the function represented in Figure 4-4.

To better understand the dependency of the inducing points on the training data, let us consider the slightly different example of learning the function in Equation 4-25 for an input window of $(x_1, x_2) \in [-2, 2]^2$ (see Figure 4-6). This window disregards an area of the input space where the function is "flat", considering now only the region near the origin where the behaviour is highly nonlinear. The results of validation for this case are shown in Figures 4-7. The performance of the FITC approximation is now significantly improved, and the method starts to outperform after around 25 inducing points. This can be attributed to the higher average quality of the randomized inducing point locations, which now are not falling in the

flatter regions that are less informative about the function.

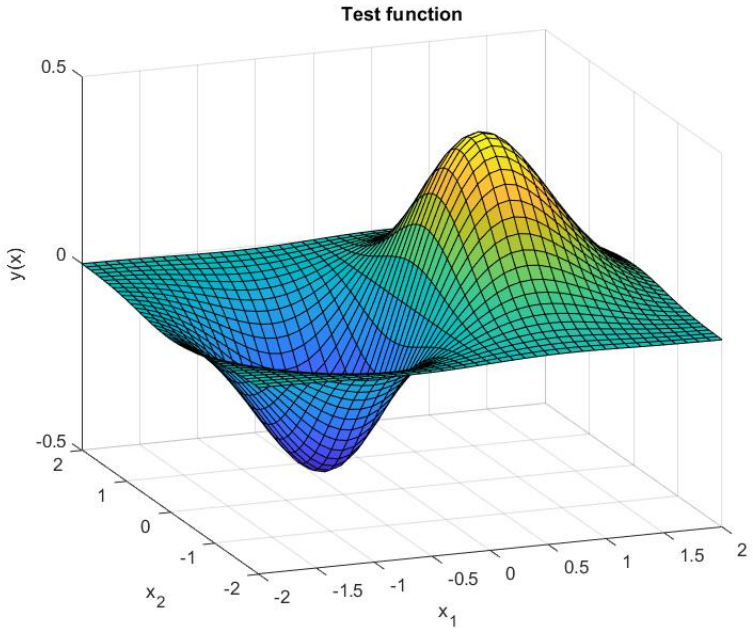

Figure 4-6: Graphical representation of the function in Equation 4-25 for $(x_1, x_2) \in [-2, 2]^2$.

The conclusion is that the SS approximation is independent of the training data, more robust to stochastic changes[4] and easier to implement. The FITC method is dependent on the training data and inducing point selection criteria, and therefore less robust. For "good" inducing point locations, the FITC approximation proved to outperform (even when the RMSE values were higher, the minimal validation error for the set of 50 iterations typically belonged to the FITC approximation).

The problem of choosing inducing point locations sometimes can be solved in clever ways. For example, the author of [28] chooses the inducing inputs to coincide with the reference trajectory coming from an MPC online application (a technique called *transduction* [41]). However, since this is not applicable to this thesis' test results in later Chapters, the remainder of this thesis will consider mostly the SS approximation as a way to achieve a computationally sparse model. This is because the model is efficient, simple to implement, it has a uniform convergence (see [49]) and it is robust to changes in the training data set. It also avoids the problem of choosing inducing point locations. Some of the shortcomings of the SS approximation, such as a difficulty of dealing with different hyper-parameter length-scales across the input space or inability to cope with non-stationary kernels, were not addressed in this Section because they are not overly restrictive for the test case examples considered later on in Chapter 6.

[4]Keeping in mind that in all of the above test cases there were ten collected sets of sampled frequencies, and the best one was chosen based on the value of the modified marginal likelihood in Equation 4-23.

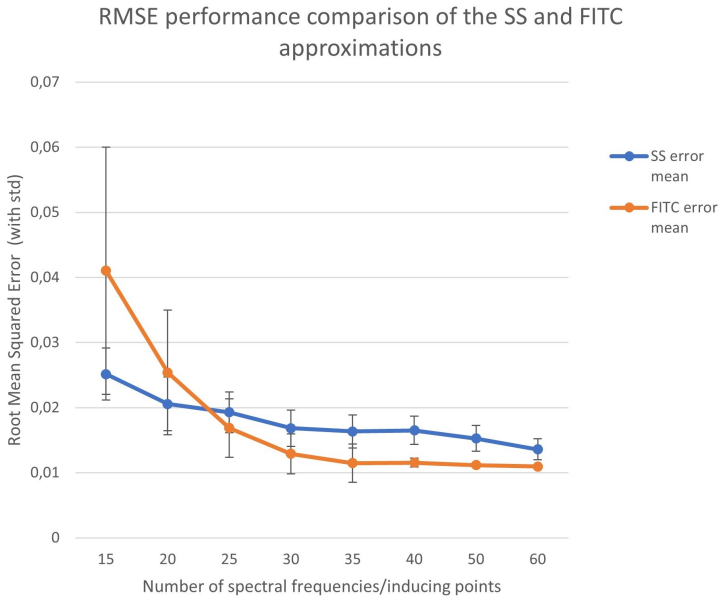

Figure 4-7: Validation results for a comparison of the FITC and SS approximations for the function represented in Figure 4-6.

Chapter 5

Model Predictive Control

The previous Chapters have dealt with obtaining a model for the system by means of collecting input-output (noisy) data and performing Identification on that data. This Chapter deals with ways to perform feedback control with the obtained model. The control strategy chosen to achieve this is Model Predictive Control (MPC).

The Chapter is organized as follows: Section 5-1 introduces the theory behind the control method; Section 5-2 introduces the adaptations to the method such that it can handle the probabilistic nature of the predictions made by the Gaussian Process (GP) model; Section 5-3 justifies the choice of control strategy and compares it with other possibilities presented in the current literature; Section 5-4 incorporates all the different ideas into a Data-driven control algorithm; Section 5-4-1 discusses certain properties of the aforementioned algorithm, design decisions and implementation challenges.

5-1 Theoretical background

MPC is a Model-based feedback control strategy that chooses the next input in an optimal way by solving an optimization problem over a finite time horizon[1]. This optimization problem is subject to constraints and it attempts to minimize a cost function specified by the user, in a way that reflects the closed-loop performance goals.

A standard MPC problem[2] has the following structure

$$\min_u J(x,u) = \min_u l_N(\boldsymbol{x}_N) + \sum_{i=0}^{N-1} l_i(\boldsymbol{x}_i, \boldsymbol{u}_i) \tag{5-1}$$

[1] This thesis makes use of the optimization software CasADi (see [54]).

[2] One can make a distinction between Linear MPC and the more general Nonlinear MPC. This thesis will mainly focus on Nonlinear MPC, since it has to work with a nonlinear dynamic model. To simplify semantics, from now on the term MPC will be used to refer to Nonlinear MPC.

subject to

$$\boldsymbol{x}_{i+1} = f(\boldsymbol{x}_i, \boldsymbol{u}_i), \quad i \in \{0, ..., N-1\} \tag{5-2}$$
$$\boldsymbol{x}_i \in \mathbb{X}, \quad i \in \{0, ..., N-1\} \tag{5-3}$$
$$\boldsymbol{u}_i \in \mathbb{U}, \quad i \in \{0, ..., N-1\} \tag{5-4}$$
$$\boldsymbol{x}_0 = \boldsymbol{y}(k) \tag{5-5}$$

Where $x = \begin{bmatrix} \boldsymbol{x}_0 & \boldsymbol{x}_1 & ... & \boldsymbol{x}_N \end{bmatrix}$ and $u = \begin{bmatrix} \boldsymbol{u}_0 & \boldsymbol{u}_1 & ... & \boldsymbol{u}_{N-1} \end{bmatrix}$. The main components of an MPC problem are:

- dynamic model: Equation 5-2 shows a generic dynamic model Equation, where the next state of the system is a function of the previous state and the input action. This model is in general discretized over a fixed time step, and each time step in the event horizon will result in another constraint for the optimization problem.
- prediction horizon: The optimal problem is solved over a predictive horizon given by a fixed number of time steps N. This value represents the amount of time steps into the future that the controller will consider in the optimization problem.
- cost function: Equation 5-1 shows a general cost function $J(x, u)$, which implicitly contains the specifications for the control performance. In general, the system behaves such that it follows a reference trajectory $X^{ref} = \begin{bmatrix} \boldsymbol{x}_0^{ref} & \boldsymbol{x}_1^{ref} & ... & \boldsymbol{x}_N^{ref} \end{bmatrix}$. A typical cost function is one that minimizes the distance between the predicted states and the reference states, such as:

$$J(x, u) = \left\| \boldsymbol{x}_N - \boldsymbol{x}_N^{ref} \right\|_P^2 + \sum_{i=0}^{N-1} \left\| \boldsymbol{x}_i - \boldsymbol{x}_i^{ref} \right\|_Q^2 + \|\boldsymbol{u}_i\|_R^2 \tag{5-6}$$

  where the notation $\|\boldsymbol{x}_i\|_P = \sqrt{\boldsymbol{x}_i^T P \boldsymbol{x}_i}$ is the Euclidean squared distance metric. The variables P, Q and R are weights which can be used to regularize the states and to work as penalty terms on the deviation from the reference trajectory. In Equation 5-6 there is a penalty on each time step for the deviation between the predicted state and the reference trajectory, as well as a penalty on the energy of the inputs u_i. More generally the inputs can also have a reference trajectory $U^{ref} = \begin{bmatrix} \boldsymbol{u}_0^{ref} & \boldsymbol{u}_1^{ref} & ... & \boldsymbol{u}_{N-1}^{ref} \end{bmatrix}$.
- Equations 5-3 - 5-4 introduce hard constraints on the input and output spaces, forcing them to stay inside a closed set of permissible values $\mathbb{X}$ and $\mathbb{U}$ respectively.
- Equation 5-5 fixes the initial state to a value $x(0)$, which can be i.e. the (noisy) measured output value coming from the sensors at a given time step k.

The closed-loop control scheme can be seen in Figure 5-1 and a visual representation of the moving time horizon concept is provided in Figure 5-2. At each time step, the MPC will solve an optimization problem to obtain an optimal control solution u over the prediction horizon. Then, only the input solution corresponding to the first time step is implemented. In the next iteration the problem is again initialized and the process is continuously iterated.

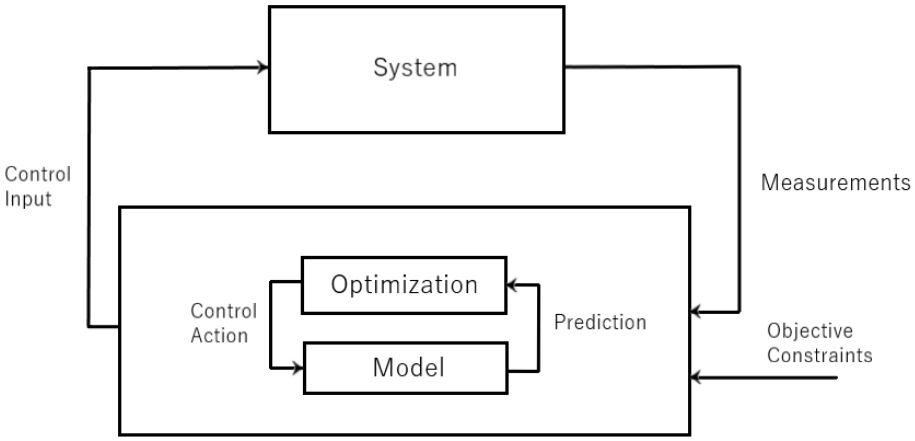

Figure 5-1: Block diagram of the MPC

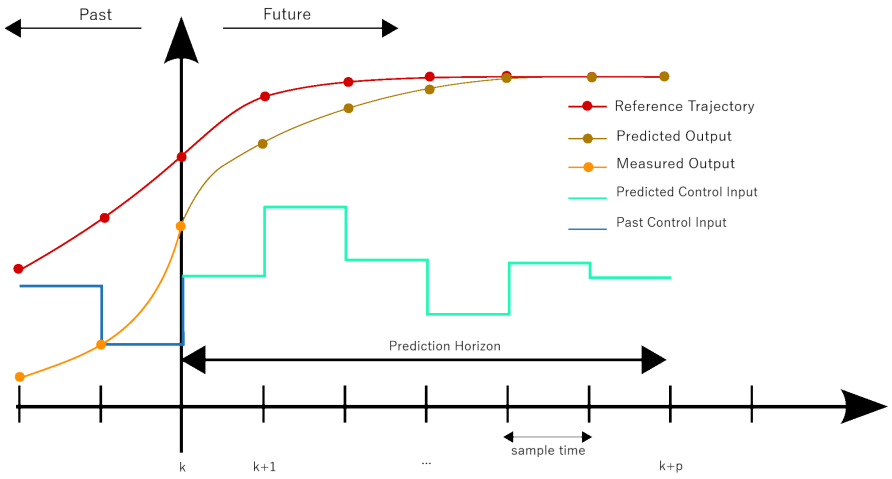

Figure 5-2: Visual depiction of the result of one iteration of the MPC

5-2 Stochastic MPC

The MPC framework described in the previous section deals with deterministic models. Since the goal is to use a probabilistic model, the optimization problem needs to be changed in order to deal with such models. In particular, since the states are probabilistic Gaussian then they do not assume discrete values, and they are also theoretically unbounded. In this case, the constraints on the optimization problem can only be guaranteed up to a certain probability, thus introducing the Stochastic MPC framework.

The states and inputs will now be Gaussian distributed stochastic variables with $\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{\mu}_i^x, \Sigma_i^x)$ and $\boldsymbol{u}_i \sim \mathcal{N}(\boldsymbol{\mu}_i^u, \Sigma_i^u)$.

5-2-1 Cost function

The cost function can be adapted by taking the expected value over the previously considered cost function in Equation 5-6:

$$J(x,u) = \mathbb{E}\left[\left\|\boldsymbol{x}_N - \boldsymbol{x}_N^{ref}\right\|_P^2 + \sum_{i=0}^{N-1}\left\|\boldsymbol{x}_i - \boldsymbol{x}_i^{ref}\right\|_Q^2 + \|\boldsymbol{u}_i\|_R^2\right] \tag{5-7}$$

$$= \left\|\boldsymbol{\mu}_N^x - \boldsymbol{x}_N^{ref}\right\|_P^2 + \mathrm{tr}(P\Sigma_N^x) + \sum_{i=0}^{N-1}\left\|\boldsymbol{\mu}_i^x - \boldsymbol{x}_i^{ref}\right\|_Q^2 + \|\boldsymbol{\mu}_i^u\|_R^2 + \mathrm{tr}(Q\Sigma_i^x + R\Sigma_i^u) \ . \tag{5-8}$$

This cost function is used by i.e. the author of [55]. Other cost functions can be introduced, such as

$$J(x,u) = \left\|\boldsymbol{\mu}_N^x - \boldsymbol{x}_N^{ref}\right\|_{\tilde{P}}^2 + \sum_{i=0}^{N-1}\left\|\boldsymbol{\mu}_i^x - \boldsymbol{x}_i^{ref}\right\|_{\tilde{Q}}^2 + \|\boldsymbol{\mu}_i^u\|_{\tilde{R}}^2 \tag{5-9}$$

$$\tilde{P} = \mathrm{tr}(P\Sigma_N^x) \tag{5-10}$$

$$\tilde{Q} = \mathrm{tr}(Q\Sigma_i^x) \tag{5-11}$$

$$\tilde{R} = \mathrm{tr}(R\Sigma_i^u) \ . \tag{5-12}$$

In Equation 5-9 the variance is not considered as an added penalty but rather as a weight in the distance between the states and the reference. This cost function is used in this work over the one in Equation 5-7, since it will do a better job at prioritizing trajectories whose mean prediction is equal to the reference states (since there is no added penalty term on the variance). A modification to this cost function similar to the one presented by the author of [27] is the following

$$J(x,u) = 1 - \exp\left(-\left\|\boldsymbol{\mu}_N^x - \boldsymbol{x}_N^{ref}\right\|_{\tilde{P}}^2\right) + \sum_{i=0}^{N-1} 1 - \exp\left(-\left\|\boldsymbol{\mu}_i^x - \boldsymbol{x}_i^{ref}\right\|_{\tilde{Q}}^2\right) \ . \tag{5-13}$$

This cost function uses an exponentially decaying term to saturate the cost. This is helpful because often the optimal solution will lead to overshooting in an attempt to push the states to the reference values as fast as possible, since the early state values have the highest penalty associated to them. Therefore, the cost function in Equation 5-13 will become a solution to overshooting.

5-2-2 Model constraints

The multiple-step-ahead prediction constraints are now given by the Gaussian process and the input and state values are stochastic Gaussian variables:

$$x_{i+1}^{j} = f(\boldsymbol{x}_i, \boldsymbol{u}_i) \sim \mathcal{N}(\mu_{i+1}^{x^j}, \Sigma_{i+1}^{x^j}) \tag{5-14}$$

where the superscript j is a reminder that each state has an associated GP prediction. Since the inputs to the GP are stochastic, the methods from Section 3-2 will apply and the mean and covariance values of the next prediction will be given by Equations 3-26 in order to propagate the trajectory uncertainty in a more realistic way. This thesis favours the Taylor approximation over the Exact Moment Matching approximation because its computational simplicity saves time during online implementations.

The constraints on the input space and the state space given previously by Equations 5-3 - 5-4 have to be changed as well. There are two ways of approaching it: either the constraints are enforced over the mean values:

$$\mathbb{E}(\boldsymbol{x}) = \boldsymbol{\mu}_i^x \in \mathbb{X} \tag{5-15}$$
$$\mathbb{E}(\boldsymbol{u}) = \boldsymbol{\mu}_i^u \in \mathbb{U} \tag{5-16}$$

or the constraints are satisfied in a probabilistic way:

$$\Pr(\boldsymbol{x} \in \mathbb{X}) \geq 1 - \alpha_x \tag{5-17}$$
$$\Pr(\boldsymbol{u} \in \mathbb{U}) \geq 1 - \alpha_u \quad . \tag{5-18}$$

This work will consider the constraints given by the first scenario. This is because the test cases considered here will not require hard constraints on the state space, but only in the input space. However, the inputs to the systems will be considered deterministic since they are chosen by the controller. Deterministic inputs still fit the framework presented above since they are just stochastic measures with a zero variance value, which is a probabilistic Dirac distribution. An example of an application that requires probabilistic state constraints is given by i.e. [55] or [28].

5-3 Controller Justification

Using a Nonlinear model is a severe limitation to the possible choices of control strategy. Much of the analysis performed in the frequency domain and stability guarantees come from the assumption of model linearity. Therefore, methods of controlling nonlinear and stochastic models are limited.

MPC is a sensible choice because it can be adapted to deal with a GP model, and also because it allows hard constraints on the input and output, as well as soft constraints in the form of penalties. MPC will have tuning parameters in the form of weighting penalties, number of horizon steps and even choice of cost function, but these can also be adjusted with minimal

knowledge of the underlying system by analysing the simulation performance.

Alternative control strategies could be Internal Model Control (IMC) or Feedback Linearization. IMC is a model based algorithm that implements a control action based on the inverse response of the model. Feedback Linearization an algorithm that fits the nonlinear model into a specific structure and then computes an input with the goal of linearizing the closed-loop input-output response. In-depth overviews of these control strategies can be found in [56] and [57] respectively. While some work has been developed in the literature incorporating GP models with these control strategies (see e.g. [58], [59]), both impose more restrictions to the model structure and less flexibility than MPC. Not only that, but neither strategy can handle hard input and output constraints. For these reasons, MPC was the controller of choice.

5-4 Data-driven MPC algorithm

The full algorithm is shown in Figure 5-3. The idea is to begin by collecting input-output information about the system to acquire training data. This training data is used to optimize over the GP hyper-parameters using the Squared Exponential (SE) Automatic Relevance Determination (ARD) kernel introduced in Section 2-2-1[3]. There is the option of sparsifying the GP predictions, which is done by applying the SS-GP method from Section 4-3.

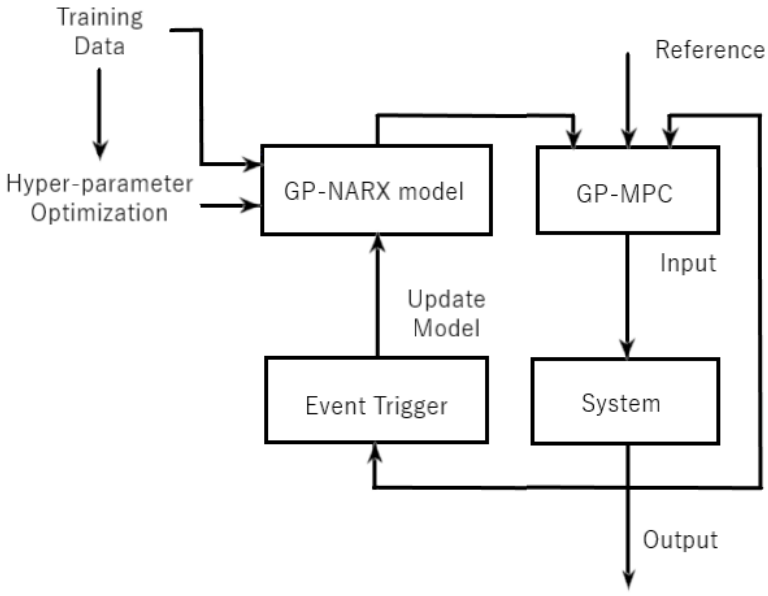

Figure 5-3: Data-driven GP-MPC algorithm

[3]There is also the option of choosing the hyper-parameters through knowledge of the system or statistical properties present in the data set.

While this step is not necessary it will greatly reduce the computational time, as will be shown later on in Chapter 6. This becomes relevant for systems whose controller sampling times need to be low (in the order of $10^{-2} - 10^{0}$ seconds) or whose dynamics require a relatively large set of training data (in the order of 10^{3} points or more).

After the training process, the GP prediction model can be fit into the MPC framework described in Section 5-2. The controller will work in closed-loop and model updates can be performed by adding points to the training data that is stored in the memory. This is possible since the GP model will always refer to the training data at prediction time, as is shown in the Equations 2-4 - 2-6. Model updates can be done in real time at every consecutive time step, or they can be triggered by some decision variable. This work follows the idea from the author of [17] where model updates are carried out by looking at the predictive variance in the previous time step, which is a natural indicator of the model's certainty at predicting in that region of the input space.

5-4-1 Implementation challenges

Notice that there are two distinct phases of the algorithm: the offline data collection and training process; and the online closed-loop control. The training process has the major goal of initializing good hyper-parameters for the GP. Since this phase is carried out offline, it is possible to collect a relatively larger amount of training points in order to obtain good hyper-parameter values, and then choose to maintain only a subset of those points to be stored in memory for the online prediction phase. This is in fact recommended, as the kernel hyper-parameters are extremely important for the quality of the model, and they will remain unchanged throughout the online controller phase. It is of course possible to perform occasional online re-training and update the hyper-parameters in real time, possibly conducting the computations in parallel with the closed-loop control performance, but this is not implemented in this work.

In addition to initializing the kernel hyper-parameters, there is the issue of choosing the appropriate regressors to use in the GP-NARX model. While this is a problem common to all System Identification techniques, by using a GP model it's possible to use the ARD kernel length-scales to determine the relevant variables to use for prediction, as explained in Section 2-2-1. However, knowledge of the underlying system is often still required in order to obtain acceptable validation results, as will be shown in Chapter 6.

The online phase requires a reference trajectory, tuning of the MPC weighting parameters, and also tuning of the event-triggered model updates, which in this work are given by the variance of the one-step-ahead prediction given by the GP. If the variance is higher than a certain threshold, that will trigger the update. The value of this threshold parameter is important to determine the regularity of model updates. It tends to be sensible to value changes and not intuitive to choose. In practice, it requires some hand tuning to ensure that model updates happen only when reaching under-explored regions of the input space. Another option is to set it to zero and have a model that always considers a past window of points, updating at every iteration. This will be further analysed in the next Chapter.

Chapter 6

Test cases and experimental results

After introducing the theoretical background in Chapters 2 - 4 and proposing a Data-driven algorithm in Chapter 5, this Chapter evaluates the performance of said algorithm through a few test cases and simulations of different systems, as well as a real-life experiment. This serves to validate the effectiveness of the method, as well as to look at some performance characteristics and to compare it with other control strategies and Data-driven methods that have been published. The Chapter begins with a test case simulation of a modified double integrator system, which is used as a 1-dimensional example to visualize the effects of a few properties mentioned in previous Chapters, such as the difference between prediction using a deterministic model and a probabilistic model. Next, the algorithm is applied to the Temperature Control Lab (TCL) setup, both in computer simulation and in the physical Arduino setup. The final experiment deals with a simulation of the Reaction Wheel Pendulum (RWP) setup in the TU Delft laboratory.

6-1 Modified Double Integrator System

The first test case deals with a modified double integrator system. A typical double integrator is a system that moves in a 1-dimensional space (for example, a particle that moves through space in a straight line). The goal is to control this position and guide it to a desired reference. The input actuation is the system's acceleration. Therefore, the system is Single Input Single Output (SISO) and it is described by one input:

- u: acceleration of the object.

and one output:

- q: 1-dimensional position of the object.

the dynamic equations of a double integrator system are the following:

$$\ddot{q}(t) = u(t) \tag{6-1}$$
$$y(t) = q(t) + \epsilon(t) \qquad \epsilon(t) \sim \mathcal{N}(0, \sigma) \ . \tag{6-2}$$

In order to make the test case more complex and take advantage of the capability of the Gaussian Process (GP) to handle nonlinear dynamics, the acceleration in Equation 6-2 is changed to be an (arbitrary) nonlinear function of the input:

$$\ddot{q}(t) = 0.1\,(\tanh(u) + \sin(4u)) \tag{6-3}$$
$$y(t) = q(t) + \epsilon(t) \qquad \epsilon(t) \sim \mathcal{N}(0, \sigma) \ . \tag{6-4}$$

Equations 6-3 - 6-4 describe the modified double integrator system. The goal of this experiment is to validate the GP-MPC as a functioning algorithm, to test the cost functions introduced in Section 5-2-1, and to conclude on the scalability of the algorithm in terms of the size of the training data set.

6-1-1 System Identification

The first step is to perform System Identification in an attempt to learn the "unknown" function from batch input/output data[1]. The batch will consist of 50 data points. The GP will model the dynamics of the acceleration, and the position will be estimated through finite differences. This is done on purpose, since it will quickly increase the variance of the solution trajectories, which will lead to more informative conclusions.

After collecting the batch data and optimizing the hyper-parameters in accordance with Section 2-3, the predicted dynamics for the full GP and the Sparse Spectrum (SS) approximation from Section 4-3 are shown in Figure 6-1.

A few notes on the results from Figure 6-1 are:

- The dynamic mapping that is learned corresponds to the acceleration, and the position and velocity will be estimated from the GP predictions through finite difference methods.
- The training points are densely placed in certain regions of the input space, but there are certain regions ($u \in [-3, -2] \cup [-1, 0] \cup [2, 3]$) where there are no training points, which reflects in the higher values of predictive variance.

[1] In this experiment, the output data set considers that the (noise corrupted) acceleration is directly observed, and the estimated dynamics are those of the acceleration. While this is unrealistic, the following test cases will not follow this methodology and will consider Nonlinear AutoRegressive eXogenous (NARX) models instead. In any case, one can interpret the acceleration values as obtained from finite differences of the measured positions taken with extremely small noise corruption.

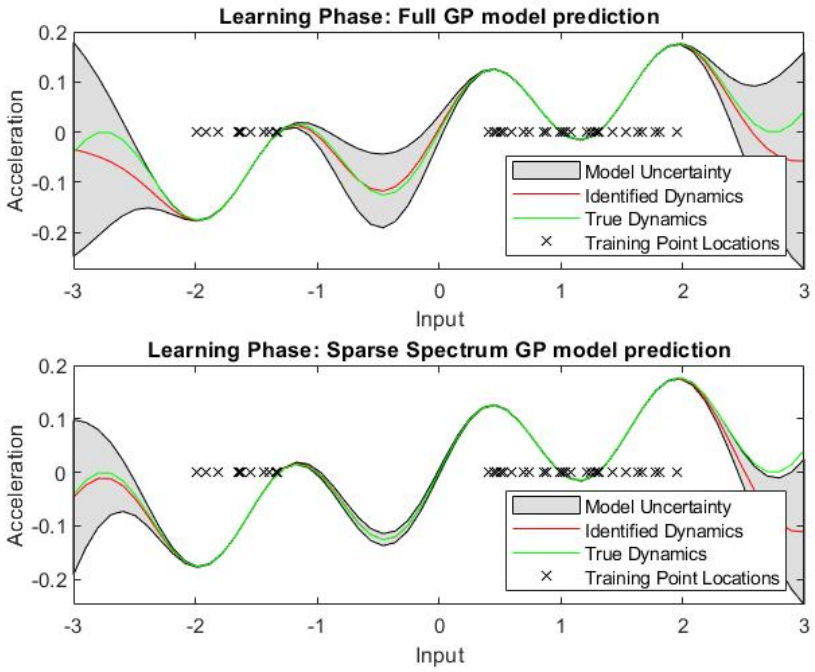

Figure 6-1: System dynamics and predicted dynamics given by a) The full GP model b) the SS-GP model with 20 frequency samples.

- The SS-GP approximation gives more confident predictions than the full GP. That is a consequence of the method not recovering the full GP even with larger amounts of spectral points (despite some of the strategies against over-fitting introduced in Section 4-3-1). Nevertheless, the mean values are accurate and the variance increases are still representative of the uncertainty of the prediction.

6-1-2 Closed-loop control

After obtaining a prediction model, the system can be controlled in close-loop using the GP-MPC structure from Section 5-2. The goal is to start at an initial position $q(0) = 3$ and to drive the system to position zero. Two different cost functions are considered: the cost functions of Equation 5-7 and 5-9. The first one does not include the variance of the resulting trajectory as a penalty, but the second one does so by using it as weighting cost for the system's predicted positions over the event horizon.

The results for the solution of one iteration of the Model Predictive Control (MPC) (the initial iteration) can be seen in Figure 6-2 for the first cost function and in Figure 6-3 for the second cost function[2]. These results show that taking the variance as a penalty input will impact the solution of the optimization problem considerably, as the increase in variance over the course of the trajectory for the first case is much higher than that of the trajectory in the

[2]These results use the prediction model obtained from the SS-GP. The results from using the model of the full GP are not presented here, as they are qualitatively similar.

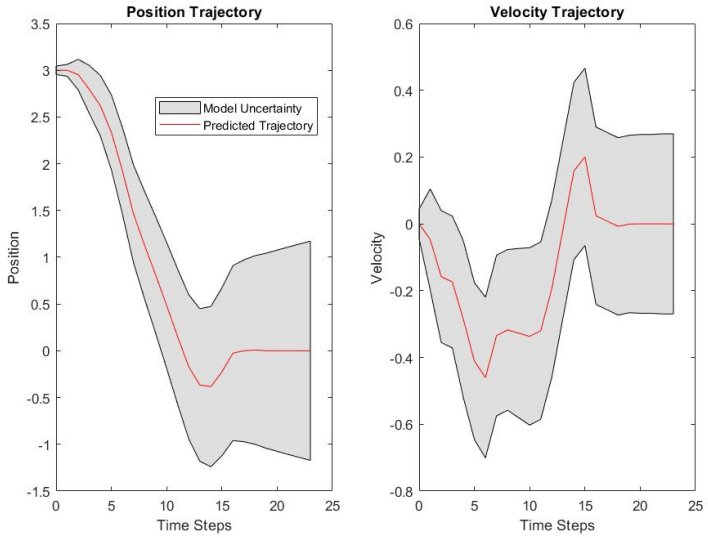

Figure 6-2: Optimal trajectory given by one iteration of the GP-MPC problem.

second case. Using a penalty on the variance value will promote using inputs that lie in the more trained regions of the input space.

This can be seen in Figures 6-4 and 6-5, which show the input locations resulting from the optimal input sequence of the MPC solution. The shaded red regions mark the areas of the input space where the predicted variance is higher, indicating higher predictive uncertainty. It is visible that the second case, where there is a penalty associated with the variance term, actively attempts to avoid these regions of uncertainty even at the cost of choosing a rougher trajectory.

6-1-3 Computational costs

From the theoretical background given in Chapters 2 and 4, it is expected that the GP-MPC algorithm may suffer from limitations in terms of computational costs. This will happen for several reasons:

1. The algorithm has to solve a nonlinear non-convex optimization problem at each time step.

2. Considering stochastic Gaussian variables adds more equality constraints to the optimization problem, as the variance term will include as many extra constraints as there are steps in the event horizon.

3. The increase in the size of the training data set for the GP will increase the time it takes to make a one-step-ahead prediction with the model (see Chapter 4). There is also a further increase in computational costs by calculating the mean function gradients, which are required for uncertainty propagation (see Section 3-2).

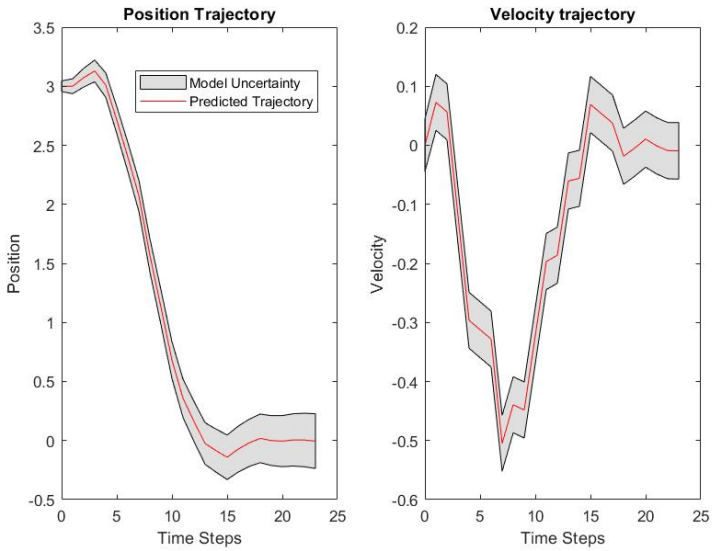

Figure 6-3: Optimal trajectory given by one iteration of the GP-MPC problem. A penalty was applied to the state variances according to the cost function in Equation 5-9

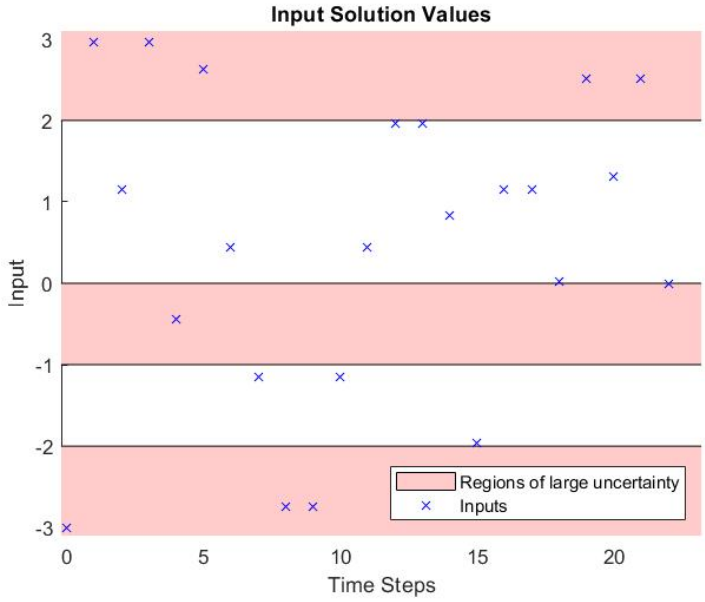

Figure 6-4: Optimal input sequence given by one iteration of the GP-MPC problem. No penalty was applied to the state variances

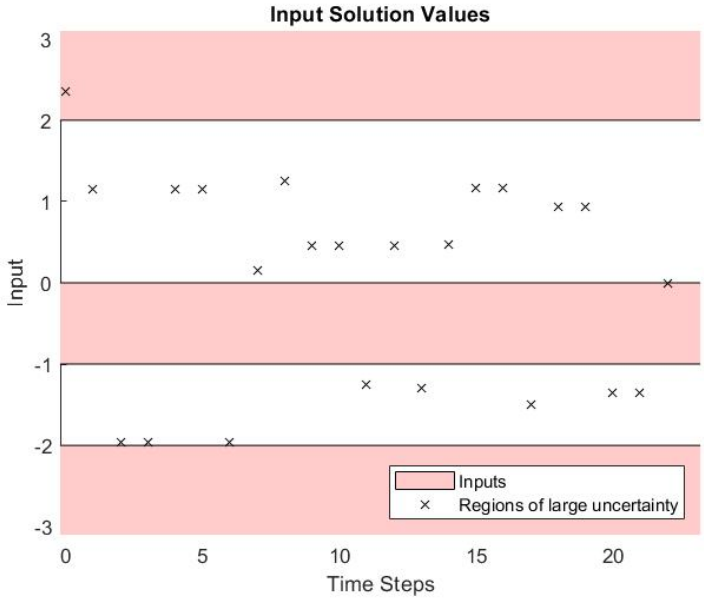

Figure 6-5: Optimal input sequence given by one iteration of the GP-MPC problem. A penalty was applied to the state variances according to the cost function in Equation 5-9

The first item concerns the limitations with nonlinear optimization methods. Nonlinear non-convex optimization is less robust and will only guarantee convergence to local minima. Not only that, but the solving time is dependent on the shape of the nonlinear cost function and the quality of the prediction will depend on good initialization criteria (see [60]). A careful study of the properties of these nonlinear optimization methods is outside of the scope of this thesis, but it is advisable to be aware of the limitations of non-convex optimization methods that may prevent the algorithm to give acceptable results in practice.

The second item is a direct consequence of using the full information available through the GP predictor. It is always possible to use only the mean function of the GP and treat the predictions as deterministic values. This is recommended if there is no serious motivation towards promoting safe trajectories or if the computational time of solving an iteration of the MPC problem is poses serious problems (for systems with controller sampling times in the order of 10^{-1} seconds or lower this could be the case).

The third item is a consequence of the computational complexity in making predictions with the GP. This issue was addressed in Chapter 4, and using sparse methods is crucial to mitigate the impact of scaling the training set, which will in turn allow the method to be used for more complex systems and for smaller controller sampling times.

Figure 6-6 shows the average solving time of computing an iteration of the MPC problem. The fastest results come from considering the variables as deterministic and working with just the predictive mean function. When considering stochastic variables, it is visible that the computational cost of solving the optimization problem scales considerably. This is because of the aforementioned extra constraints on the variance. In the case of the full GP, a prediction scales $\mathcal{O}(N^2)$ with the number of training points. In the case of the SS-GP approximation,

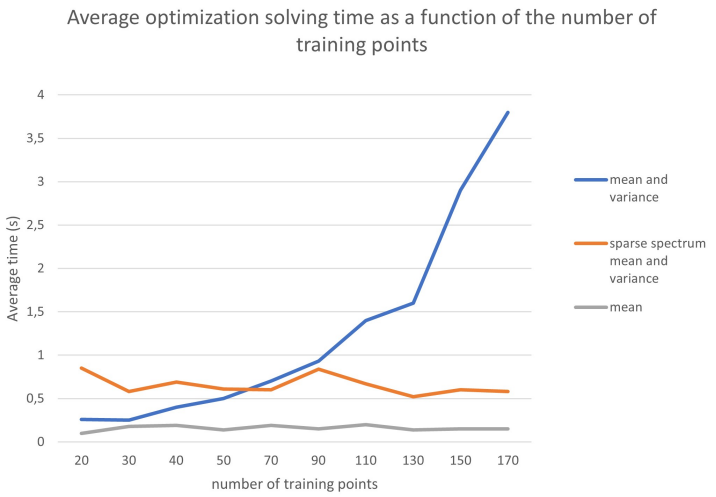

Figure 6-6: Average computational time of solving an iteration of the MPC during closed-loop control using: deterministic states (grey); stochastic states with the full GP (blue); stochastic states with the SS-GP (orange).

the prediction scales $\mathcal{O}(m^2)$, where m is the (fixed) number of spectral frequencies.

This (in)dependence on the number of training points is visible in Figure 6-6. While the average solving time increases rapidly with an increase in the training set for the case of the full GP, the sparse approximation roughly maintains the computational time constant throughout the experiment[3]. However, using sparse approximations is only advantageous past a certain threshold number of points (in this case around 60). This is because there are more computations involved in the sparse prediction, which will make it slower for a small number of training points.

From these results one can conclude on the importance of using computationally cheaper solutions in order to feasibly scale the method to more complex systems[4]. One only has to be careful at monitoring the quality of the sparse approximation, as often considering small data sets will yield bad results and numerical instabilities. However, since sparse predictions are not justified for smaller training data sets, this is typically not an issue.

6-2 Temperature Control Lab

The second test case is performed on the TCL setup[5]. The Temperature Control Lab is a small Arduino board meant as an educational application for feedback control. The kit

[3]In fact, the computational time slightly decreases, which can be justified by the fact that the overall uncertainty decreases with more points, which causes the optimal solution to converge more quickly.

[4]There are of course other considerations that were not addressed. Reducing the prediction horizon will decrease the solving time, as well as initializing the optimization problem at reasonable values such as the previous optimal solution.

[5]For further details the reader is referred to the official website at http://apmonitor.com/heat.htm

is equipped with two heaters and two temperature sensors, one placed in each heater. The physical board is shown in Figure 6-7. Each heater power output can be actuated individually through a digital controller.

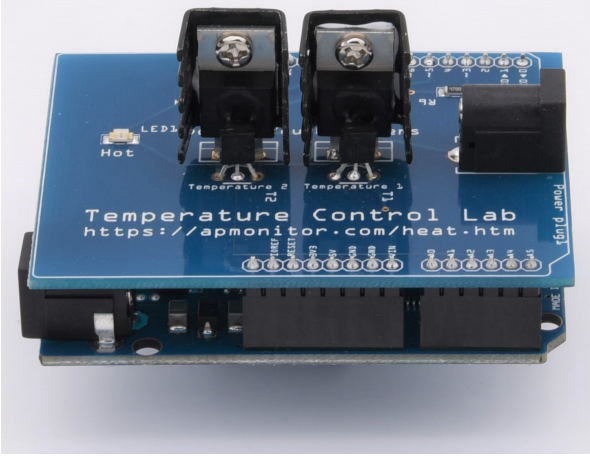

Figure 6-7: Arduino board showing two heaters and two temperature sensors.

As the heaters are provided with power, the temperature will increase and thermal energy will be traded with the surroundings through conduction, convection and radiation. The goal is to control the temperature of each individual heater and drive it to a desired reference. Therefore, the system will be described by two inputs:

- Q_1: Percentage of power actuation acting on heater 1.
- Q_2: Percentage of power actuation acting on heater 2.

and two outputs:

- T_1: Measured temperature from sensor 1.
- T_2: Measured temperature from sensor 2.

White-box dynamics for each heater can be derived from energy balance equations[6] using theory from heat transfer (see i.e. [61]). The energy balance equations are given by

$$mc_p\frac{dT_1}{dt} = UA(T_\infty - T_1) + \epsilon\sigma A(T_\infty^4 - T_1^4) + UA_s(T_2 - T_1) + \epsilon\sigma A_s(T_2^4 - T_1^4) + \alpha_1 Q_1 \quad \text{(6-5)}$$

$$mc_p\frac{dT_2}{dt} = UA(T_\infty - T_2) + \epsilon\sigma A(T_\infty^4 - T_2^4) - UA_s(T_2 - T_1) - \epsilon\sigma A_s(T_2^4 - T_1^4) + \alpha_2 Q_2 \quad . \quad \text{(6-6)}$$

The balance Equations 6-5 - 6-6 are composed of five different terms. In the case of heater 1 (heater 2 is analogous) they are:

[6]For the full derivations the user is referred to https://apmonitor.com/pdc/index.php/Main/ArduinoModeling2

Quantity	**Value**
Ambient temperature (T_∞)	296.15K (23ºC)
Heater output (Q_1)	0 to 1 W (in %)
Heater output (Q_2)	0 to 0.75 W (in %)
Heater factor (α_1)	0.01 W/(% heater)
Heater factor (α_2)	0.0075 W/(% heater)
Heat capacity (C_p)	500 J/kg-K
Surface area not between heaters (A)	$1\dot{1}0^{-3}m^2$
Surface area between heaters (A_s)	$2\dot{1}0^{-4}m^2$
Overall heat transfer coefficient (U)	10 W/m^2-K
Emissivity (ϵ)	0.9
Stefan Boltzmann constant (\sigma)	5.67 \dot 10^{-8} W/m^2-K^4

Table 6-1: Variables from the energy balance dynamic Equations 6-5-6-6

- A convection term between heater 1 and the ambient room: $Q_{C1\infty} = UA(T_\infty - T_1)$.
- A radiation term between heater 1 and the ambient room: $Q_{R1\infty} = \epsilon\sigma A(T_\infty^4 - T_1^4)$.
- A convection term between heater 1 and heater 2: $Q_{C12} = UA_s(T_2 - T_1)$.
- A radiation term between heater 1 and heater 2: $Q_{R12} = \epsilon\sigma A_s(T_2^4 - T_1^4)$.
- The power coming in from the actuator: $Q_{U1} = \alpha_1 Q_1$.

Where Table 6-1 indicates the meaning of each variable, as well as an estimate of the value according to the website.

Equations 6-5 - 6-6 will be used initially to simulate the system and apply the Data-driven control algorithm from Section 5-4. After that, the algorithm will be implemented and tested on a physical setup identical to the one in Figure 6-7.

6-2-1 System Identification

After the illustrative beginning simulations of Section 6-1, this simulation further approaches realistic scenarios by considering that the system described by Equations 6-5 - 6-6 can only be observed through noisy measurements of the true states corresponding to each of the heater temperatures

$$y_1(k) = T_1(k) + \epsilon_1(k), \qquad \epsilon_1 \sim \mathcal{N}(0, \sigma_1) \tag{6-7}$$
$$y_2(k) = T_2(k) + \epsilon_2(k), \qquad \epsilon_2 \sim \mathcal{N}(0, \sigma_2) \ . \tag{6-8}$$

Where ϵ_1 and ϵ_2 introduce Gaussian white noise. The model then becomes a NARX-GP model (as introduced in section 3-1) with the following regressors

$$T_1(k+1) = f(\boldsymbol{y}(k), \boldsymbol{u}(k)) = f(y_1(k), y_2(k), Q_1(k), Q_2(k)) \tag{6-9}$$
$$T_2(k+1) = f(\boldsymbol{y}(k), \boldsymbol{u}(k)) = f(y_1(k), y_2(k), Q_1(k), Q_2(k)) \ . \tag{6-10}$$

The choice of regressors in this case is trivial because of the underlying knowledge of the system dynamics. It corresponds to considering a NARX model with a one-step-behind window of inputs and outputs of the system at the previous time step.

For the simulation experiments it was considered that $\sigma_1 = 0.1$ and $\sigma_3 = 0.3$. The first step is to collect input-output data which will be used to train the GP hyper-parameters offline. While it's not clear how to choose the inputs for Identification such that the system is excited in an appropriate way for the training phase, the choice was to use a pseudo-random sequence of inputs for each of the heaters, acting over 300 time steps. The controller sampling time was chosen to be five seconds[7], which corresponds to a simulated experiment of approximately 25 minutes. Since during the online phase the controller will be performing model updates as the system moves through lesser trained regions of the input space there is no need for the training data to be equally spread over the input space, so long as the training data enables learning of proper values for the kernel hyper-parameters.

Since there is direct access to the system dynamics, model validation can be performed after training by sampling data at random and comparing the model prediction with the true system response. Because this system is Multiple Input Multiple Output (MIMO), validation results cannot be plotted and carried out visually (as was done on Section 6-1). Instead, validation is done by using the Mean Squared Error (MSE) and Mean Standardized Log Loss (MSLL) metrics introduced in Section 2-4.

Table 6-2 shows the values of the validation results used to assess the quality of the model.

	MSE		**MSLL**	
nº points	full GP	SS-GP	full GP	SS-GP
20	0.06	0.07	-3.1	-2.2
	0.02	0.02	0.6	0.7
40	0.007	0.007	-4.8	-4.8
	0.004	0.004	-5.9	-5.9
70	0.002	0.002	-6.9	-6.9
	0.003	0.003	-7.5	-7.2
120	0.001	0.001	-6.8	-6.8
	0.002	0.003	-7.1	-5.9

Table 6-2: Validation data for the prediction model.

The results show that the model can accurately predict the system dynamics even with a relatively small number of data points. However, numerical stability of the approximation

[7] The initial controller sampling time of one second resulted in some numerical problems when applied to the physical setup, hence the increase. The model was not able to capture the slow-moving dynamics of the temperature system from the one-step-behind regression window.

and an ill-conditioned kernel matrix can happen as the data set gets smaller. An initial data set of 50 points will consistently give good predictive results for the initial model.

Since the predictions from the model will be iterated, it's important to also propagate the system uncertainty, so that the model does not underestimate the deviations from the mean trajectory. Figures 6-8 and 6-9 showcase the importance of the propagation of uncertainty methods of Section 3-2 rather than taking the naive approach. Notice that Figure 6-8 is overly optimistic on the predicted trajectory, and the real trajectory ends up outside the confidence region. Figure Figures 6-8 predicts more conservatively in a realistic way by accounting the uncertainty coming in from previous inputs.
The Figures also shows that the predictor gives reasonable trajectories for the system's behaviour even after 15 time steps.

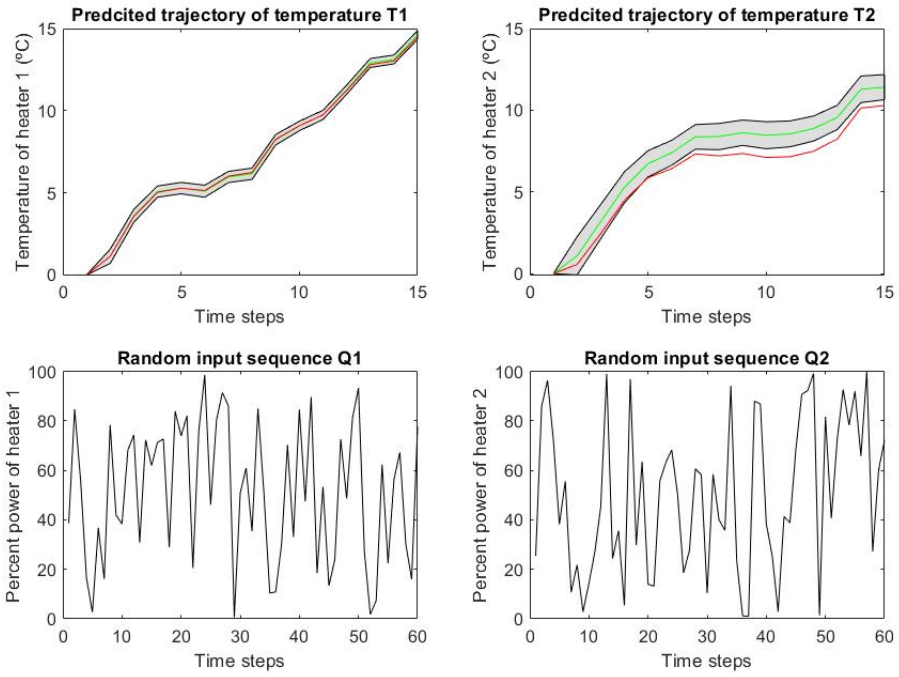

Figure 6-8: Predicted trajectory (green) and true system trajectory (red) for a pseudo-random sequence of inputs, without accounting for noise in the regressors

6-2-2 Closed-loop results

After training the kernel hyper-parameters and performing the validation steps, it's straightforward to use the predictor model to implement the closed-loop control action. The goal is for each of the heater temperatures to follow their respective step reference. In order to obtain good model updates online, the variable that triggers said updates based on the predictive variance needs to be properly tuned. Figure 6-10 shows the test results.

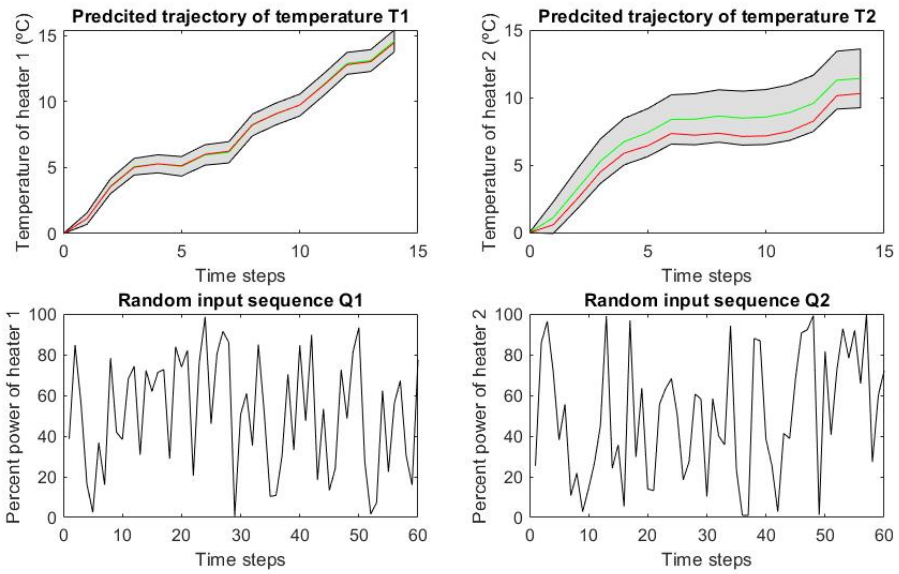

Figure 6-9: Predicted trajectory (green) and true system trajectory (red) for a pseudo-random sequence of inputs, accounting for noise in the regressors

The results show that the algorithm manages to successfully drive the inputs to the desired reference points. However, the experiment requires iterating simulations in order to tune the free penalty terms in the MPC cost function as well as the model point updates. Using variance-triggered updates is recommended in this particular experiment because the simpler method of using a window of past inputs will result in the model losing valuable information when the output is kept at a certain constant value for a large amount of time (all the training points will become the same).

6-3 Results on the physical setup

This Section implements the algorithm in the physical Arduino board. The experimental results on the experimental setup begin similarly with the acquisition of input-output data points to train the GP. However, initial experiments show that using the same regressors as in Section 6-2-1 give poor validation results. The reason for this is that there is a delay introduced in the system given by the assumption made in simulation that the temperature of the sensors is equal to the temperature of the heaters. Instead, there should be a distinction between the temperature of the sensors (T_{S1} and T_{S2}) and the temperature of the heaters (T_{H1} and T_{H2}) to account for the heat conduction between the heater and sensor. The dynamic equations can then be written as

$$mc_p\frac{dT_{H_1}}{dt} = UA(T_\infty - T_{H_1}) + \epsilon\sigma A(T_\infty^4 - T_{H_1}^4) + UA_s(T_{H_2} - T_{H_1}) + \epsilon\sigma A_s(T_{H_2}^4 - T_{H_1}^4) + \alpha_1 Q_1 \quad (6\text{-}11)$$

$$mc_p\frac{dT_{H_2}}{dt} = UA(T_\infty - T_{H_2}) + \epsilon\sigma A(T_\infty^4 - T_{H_2}^4) - UA_s(T_{H_2} - T_{H_1}) - \epsilon\sigma A_s(T_{H_2}^4 - T_{H_1}^4) + \alpha_2 Q_2 \quad (6\text{-}12)$$

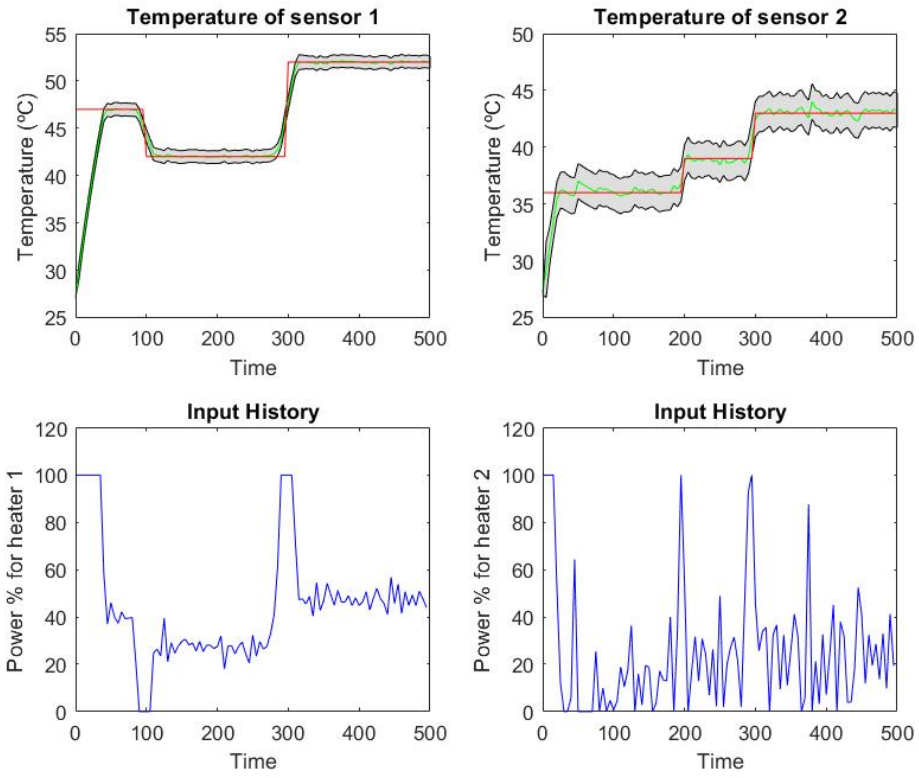

Figure 6-10: Simulation results of the closed-loop control

$$\tau \frac{dT_{C_1}}{dt} = T_{H_1} - T_{C_1} \tag{6-13}$$

$$\tau \frac{dT_{C_2}}{dt} = T_{H_2} - T_{C_2} \quad . \tag{6-14}$$

Equations 6-13 - 6-14 show the delay between heat conduction from the heater to the respective sensor.

In practice there is only access to the temperature of the sensors, which means that the system will suffer from a delay in the input-output response. As such, it's necessary to change the regressors in order to accurately predict the system dynamics. Not only that, but the slow moving nature of the system dynamics makes it difficult to excite the system in ways that are sufficiently informative to perform System Identification.

In order to account for the delay, the system regressors are changed to a larger window of past system outputs:

$$T_1(k+1) = f(y_1(k), y_2(k), y_1(k-1), y_2(k-1), Q_1(k), Q_2(k)) \tag{6-15}$$
$$T_2(k+1) = f(y_1(k), y_2(k), y_1(k-1), y_2(k-1), Q_1(k), Q_2(k)) \ . \tag{6-16}$$

After looking at the optimized length-scales of the Automatic Relevance Determination (ARD) kernel and removing regressors that are not impactful to the output prediction, the final choice of regressor structure is:

$$T_1(k+1) = f(y_1(k), y_2(k), y_1(k) - y_1(k-1), Q_1(k)) \tag{6-17}$$
$$T_2(k+1) = f(y_1(k), y_2(k), y_2(k) - y_2(k-1), Q_2(k)) \ . \tag{6-18}$$

Equations 6-17 - 6-18 map the system output as a function of the temperature of each heater at the previous time step, the respective heater power term and a term $y(k) - y(k-1)$ that is representative of the difference in temperature between the heater and the sensor.

The experimental results in closed-loop control can be seen in Figures 6-11 and 6-12. Figure 6-11 uses the variance-penalizing cost function in Equation 5-9 and Figure 6-12 uses the saturating cost function in Equation 5-13. Computational complexity was not an issue for this experiment, since the average time of solving one iteration of the MPC problem was of around 0.25 seconds, well bellow the controller sampling time of five seconds.

We can see that the algorithm still achieves the goal of driving the states to the desired reference points. The overshooting behaviour in Figure 6-11 is justified from the delay inherent to the system and the fact that the regressors may not be the most appropriate, since in the experimental case there is no true knowledge of the underlying function. Nevertheless it is visible that the algorithm works on the physical setup, and the overshooting response can partially be addressed with the use of the saturating cost function.

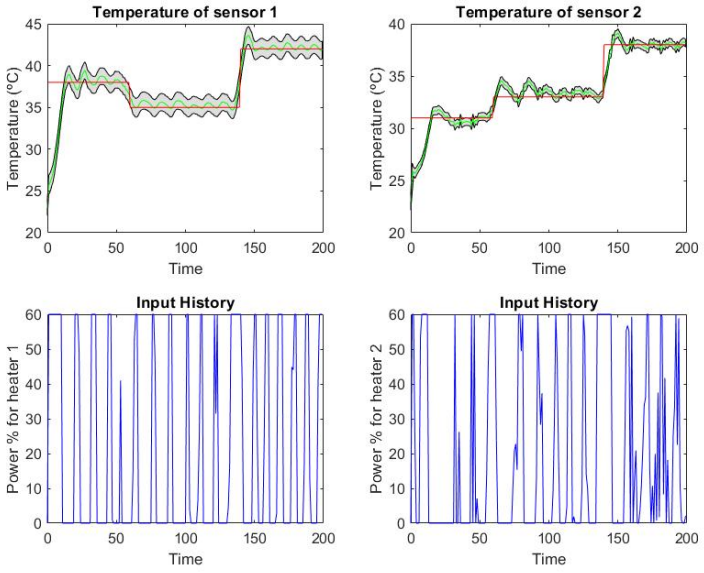

Figure 6-11: Experimental results of the TCL closed-loop control using the cost function of Equation 5-9

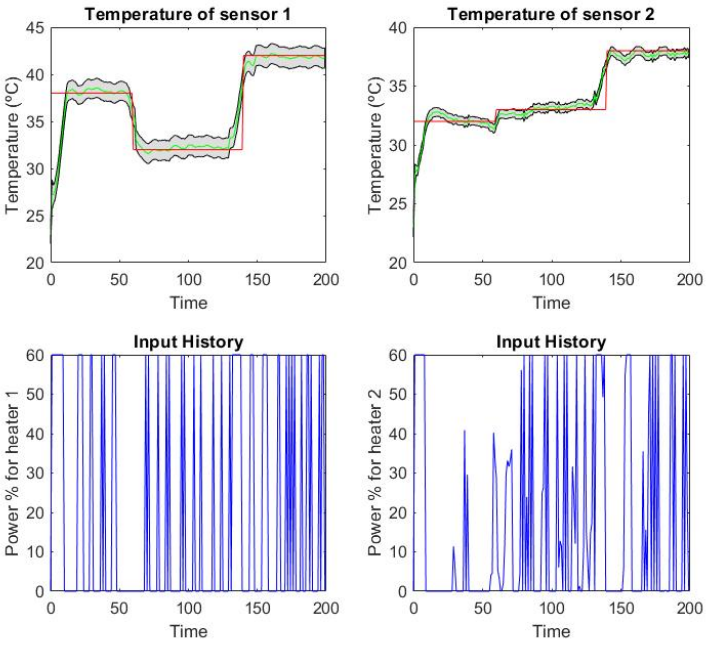

Figure 6-12: Experimental results of the TCL closed-loop control using the cost function of Equation 5-13

6-4 Reaction Wheel Pendulum

The RWP is an inverted pendulum device designed for control purposes (see Figure 6-13). The Inverted Pendulum experiment is a typical benchmark example in Control applications due to its unstable equilibrium point at the upright position. In the RWP, the actuation is achieved through a flywheel connected at the end of the rod. This flywheel is attached to a DC motor which can be controlled by an input current. To swing-up and stabilise the system in an upright position the DC motor drives the flywheel wheel forward and backward, which will make the pendulum move according to the conservation of angular momentum in the system [8].

Figure 6-13: RWP system setup. (Figure taken from the catalogue.)

The schematic diagram of the RWP system is shown in Figure 6-14. The main components of the setup are:

- The mechanical structure.
- The PWM controlled DC motor.
- Two incremental encoders, one measuring the flywheel angle (1024 imp/rpm quadrature), and one measuring the pendulum angle (5000 imp/rpm quadrature).
- Two counterweights, one fixed and one adjustable to change the center of gravity of the Pendulum.

[8] For further details on the setup and the derivation of system dynamics the reader is referred to http://www.inteco.com.pl/products/reaction-wheel-pendulum/

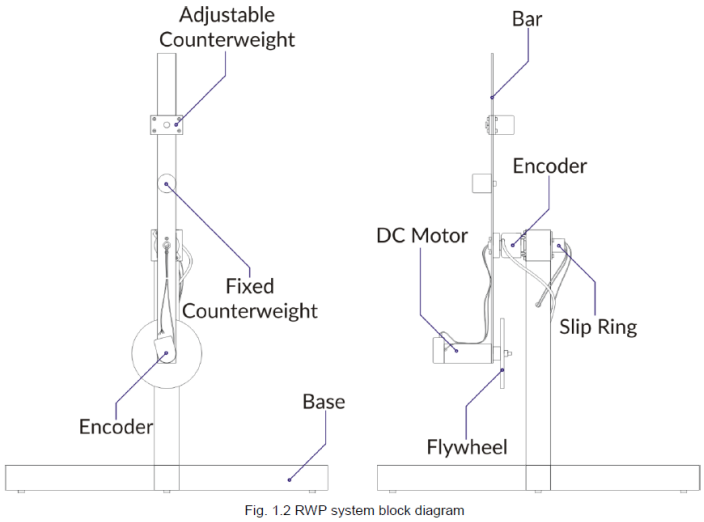

Figure 6-14: RWP system components. (Figure taken from the catalogue.)

- The power interface box containing a power supply, power amplifiers and a signal conditioning unit.

From the system specifications it's clear that the system has will have one input:

- u: Input current to the DC motor.

and two outputs:

- θ: Pendulum angle.
- ϕ: Flywheel angle.

The system dynamics are given by the following equations[9]:

$$\ddot{\theta} = \frac{MgL\sin\theta}{J_\theta} - \frac{k_t(u - k_\phi\dot{\phi})}{J_\theta R} + \frac{\mu_\phi\dot{\phi}}{J_\phi} - \frac{k_{AS}}{J_\theta}\tanh(\Gamma u) \tag{6-19}$$

$$\ddot{\phi} = \frac{k_t(u - k_\phi\dot{\phi})}{J_\phi R} - \frac{\mu_\phi\dot{\phi}}{J_\phi} - \frac{k_{DS}}{J_\phi}\tanh(\Delta u) \tag{6-20}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \theta \\ \phi \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \end{bmatrix}, \qquad \epsilon_1, \epsilon_2 \sim \mathcal{N}(0, \sigma_N) \ . \tag{6-21}$$

The physical meaning of each variable and their estimated values are provided in Table 6-3.

[9]The nonlinear term on the input u is not included in the official RWP documentation, but it is used in the dynamic simulations provided in the example MATLAB code. Therefore, they were kept for the experiments in this paper.

Parameter	Value	Units	Description
ϕ	-	rad	flywheel angle
$\dot{\phi}$	-	rad/s	flywheel angular velocity
θ	-	rad	pendulum angle
$\dot{\theta}$	-	rad/s	pendulum angle velocity
M	0.892	kg	weight of the pendulum
g	9.81	m/s^2	gravitational constant
L	0.028	m	pendulum center of mass
k_ϕ	0.027	Vs/rad	electrical constant of the DC motor
k_t	0.025	Nm/A	torque constant of the DC motor
μ_ϕ	8.82e-5	Nms	friction coefficient in the DC motor axis
μ_θ	1.32e-6	Nms	friction coefficient in the pendulum axis
J_ϕ	2.3e-4	kgm^2	moment of inertia of the DC motor and flywheel relative to the the motor shaft axis
J_θ	0.029	kgm^2	moment of inertia of the pendulum relative to its pivot point
R	2.4	Ohm	DC motor terminal resistance
k_{AS}	0.008	Nm/A	constant for DC motor nonlinearities acting on the pendulum
k_{DS}	0.003	Nm/A	constant for DC motor nonlinearities acting on the flywheel
Γ	10.64	-	constant for DC motor nonlinearities acting on the pendulum
Δ	10.00	-	constant for DC motor nonlinearities acting on the flywheel

Table 6-3: Calculated parameter estimations for the pendulum dynamics

6-4-1 System Identification

Once again the first step is to collect batch data and perform System Identification by training the kernel hyper-parameters and performing validation results. In this case, assuming that there is no knowledge about the underlying system dynamics one could begin by picking the following regressors in a NARX model:

$$y_1(t) = f(y_1(t-1), y_2(t-1), u(t-1)) + \epsilon_1(t) \tag{6-22}$$
$$y_2(t) = f(y_1(t-1), y_2(t-1), u(t-1)) + \epsilon_2(t) \tag{6-23}$$
$$\epsilon_1(t) \sim \mathcal{N}(0, \sigma_1) \tag{6-24}$$
$$\epsilon_2(t) \sim \mathcal{N}(0, \sigma_2) \ . \tag{6-25}$$

Since the model dynamics in Equations 6-19 - 6-20 depend on the angular velocities of the pendulum and reaction wheel, the previous choice of regressors will gives poor validation results. Then, one could try to extend the past window of considered inputs and outputs by an extra time step:

$$y_1(t+1) = f(y_1(t), y_2(t), y_1(t-1), y_2(t-1), u(t), u(t-1)) + \epsilon_1(t) \tag{6-26}$$
$$y_2(t+1) = f(y_1(t), y_2(t), y_1(t-1), y_2(t-1), u(t), u(t-1)) + \epsilon_2(t) \tag{6-27}$$
$$\epsilon_1(t) \sim \mathcal{N}(0, \sigma_1) \tag{6-28}$$
$$\epsilon_2(t) \sim \mathcal{N}(0, \sigma_2) \ . \tag{6-29}$$

From the optimized ARD length-scales, it is straightforward to conclude that the regressor $u(t-2)$ does not affect the system dynamics. However, this choice of regressors still will not yield good predictive results. This is because the dynamics are dependent on the velocity, which is roughly proportional to the differences $y_1(t-1) - y_1(t-2)$ and $y_2(t-1) - y_2(t-2)$. Since during simulations the pendulum flywheel angle ϕ will change significantly and in practice become unbounded, the GP prediction will always consider that it is operating in an unknown region of the input space.

To get around this problem one has to use knowledge of the system while picking the appropriate regressors[10]. A more appropriate choice of regressors considers a measure of the angular velocities by taking the difference between two consecutive outputs:

$$y_1(t) = f(y_1(t-1), y_1(t-1) - y_1(t-2), y_2(t-1) - y_2(t-2), u(t-1)) + \epsilon_1(t) \tag{6-30}$$
$$y_2(t) = f(y_2(t-1) - y_2(t-2), u(t-1)) + \epsilon_2(t) \tag{6-31}$$
$$\epsilon_1(t) \sim \mathcal{N}(0, \sigma_1) \tag{6-32}$$
$$\epsilon_2(t) \sim \mathcal{N}(0, \sigma_2) \ . \tag{6-33}$$

The mapping used in Equations 6-30 - 6-31 is the one used for the remaining of the experiments. The training data is collected by dropping the pendulum from the halfway point and applying a pseudo-random sequence of input values. This will collect around 70 points in the bottom half of the pendulum swing.

The batch data set is used to optimize the negative marginal log-likelihood over the kernel hyper-parameters. Notice the fact that no training data is collected at the top half of the pendulum motion. This should not be relevant to achieving the closed-loop swing-up and stabilization in the top position, since the algorithm is capable of adding points to the model in real time and explore the top-half region online.

The validation results can be seen in Table 6-4. The GP model is able to predict the system dynamics with sufficiently small error values, both for the full GP and for the SS-GP approximation.

[10]This problem is common to all machine learning methods, since it is still required to provide training points that are relevant and affect the change in output.

	MSE		MSLL	
nº points	full GP	SS-GP	full GP	SS-GP
20	0.005 0.15	0.005 0.17	-0.7 -1.5	0.5 -1.6
30	0.007 0.04	0.007 0.04	-0.8 -1.1	-0.6 -1.1
50	0.003 0.004	0.003 0.004	-2 -1.4	-1.9 -1.4
70	0.002 0.004	0.002 0.005	-2.3 -1.4	-2.3 -1.4
90	0.002 0.006	0.002 0.006	-2.1 -2.3	-2.0 -2.3
120	0.002 0.003	0.002 0.003	-2.0 -2.0	-2.1 -2.4
150	0.001 0.0009	0.002 0.001	-2.5 -2.5	-2.1 -2.2

Table 6-4: Validation data for the RWP pendulum.

6-5 Closed-loop results

Figure 6-15 shows the closed-loop results. The top stabilization is always achieved after a couple of attempts of swing-up, and in many cases only after one attempt at swing-up. This is despite the fact that the top half of the swing-up has no training points in the beginning of the experiment. Additional training points are collected in real time, which will improve prediction performance and allow for fast convergence to the desired reference position.[11]

The algorithm proved to be numerically stable and reliable in the solution of the iterative nonlinear optimization problem. The first iteration of the MPC takes significantly longer (around 1 second to solve) but subsequent iterations are significantly faster when initialized with the optimal solution of the previous time-step.

Implementation of the algorithm in the physical setup should follow a close methodology to the one applied in simulations. The one drawback is that the average computational time of solving one iteration of the MPC (for the test simulation) was of around 0.12 seconds, roughly the same as the controller sampling time. In practice this would result in a one-step delay which could impact the upright stabilization. Possible solutions to this problem would be to incorporate the delay into the NARX model by shifting the regressors one step backwards, or to increase controller sampling time if possible.

[11] In this experiment the state is augmented with the energy of the system, which is used as an additional variable for control. This is important because an under-actuated pendulum needs to swing in the opposite direction to build momentum, which is hard to imprint to the controller without a state augmentation. This experiment is not to be confused with the common reinforcement learning experiments (see i.e. [27]) where the controller has to converge to a swing-up policy on its own.

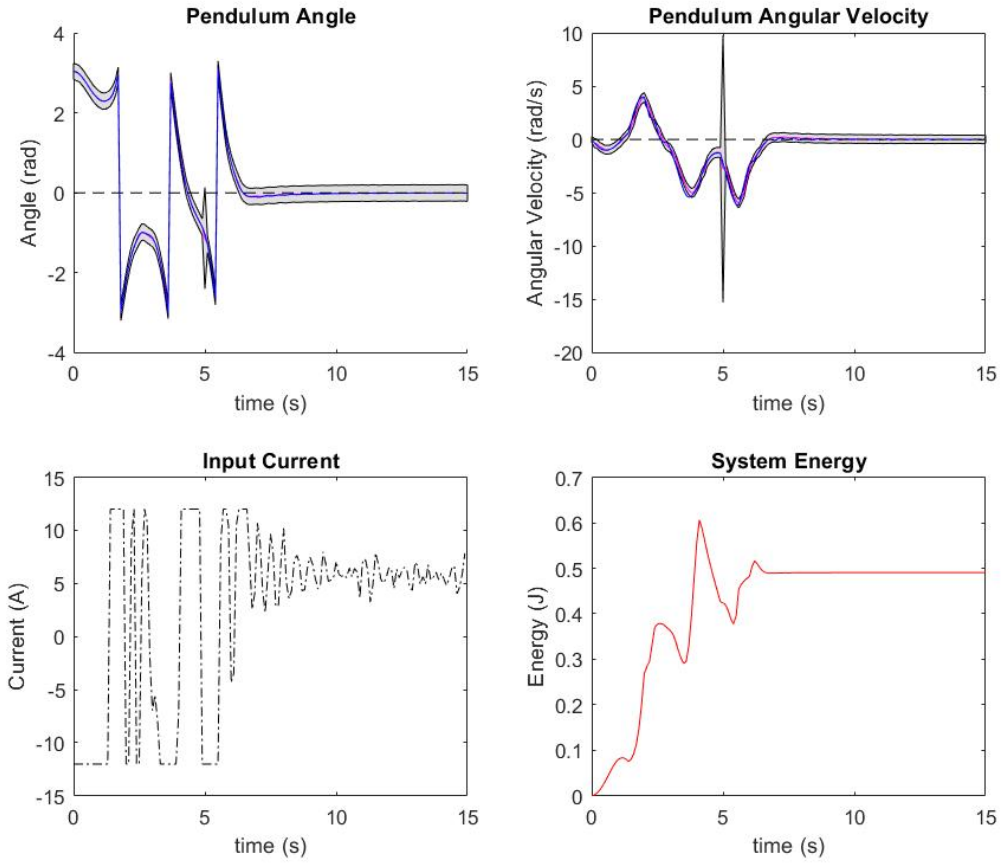

Figure 6-15: Closed-loop simulation results of the RWP swing-up and upright stabilization

Chapter 7

Conclusion

This Chapter concludes this thesis work. The Chapter is structured as follows: Section 7-1 will overview the results of this thesis and answer the questions introduced in the introductory Section 1-2. Section 7-2 will conclude by proposing some topics for future work and further research in this field.

7-1 Research Questions and Results

1. *Can a Gaussian Process (GP) be used as a model for System Identification using only batch input/output data? Does the model achieve reasonable prediction results?*

   Gaussian Process Regression (GPR) used in combination with a universal kernel is a Machine Learning method capable of successfully predicting the behaviour of a wide variety of functions. Chapters 3 - 4 introduced ways of applying the method to System Identification. Chapter 6 showed that for all test cases the learned model was able to predict the evolution of the system accurately and show good validation results with relatively small batches of training data. This suggests that GPs can be used to model a system from input/output data only, without the need to include a nominal model (which is how they are applied in most of the literature).

2. *Can the model be used in a (Nonlinear) Model Predictive Control (MPC) framework?*

   The experiments in Chapter 6 show that Nonlinear MPC can be used in conjunction with a GP model for a variety of systems. While the solutions are only guaranteed to converge to local minima, initiating the optimization problem at a random feasible point in the beginning and later using the solution at the previous time step proved to be an effective strategy in the practical cases.

3. *Compare the above mentioned control performance to MPC control using more standard deterministic models, in terms of advantages and disadvantages. Particularly, can one take advantage of useful properties unique to the Gaussian Process, such as the stochastic Gaussian predictions and the way this method makes use of the training data as a linear smoother?*

   Chapter 6 showed that the stochastic variables coming from the GP prediction could influence trajectories and promote more certain trajectories with lower variance. This is a degree of freedom that can be used in safety-critical applications where the quality of the prediction and the confidence in the outcome are of importance. Conversely, it is also possible to use the mean values of the GP predictor as deterministic in order to lower computational complexity, if there is no benefit to be gained from promoting safe trajectories for that specific application.

   It was also argued that storing the training data in memory was helpful in terms of quickly updating the model online, and that these online updates could once again be based on the variance metric that is exclusive to the GP model. Since a linear smoother approximates patterns in the data through direct access to the training data set, it is easier to update the model online by adding or removing data points from the set.

4. *What are the potential barriers in implementation, such as the performance dependence on the training data, the required prior knowledge and the ability to scale the problem to more complex systems and larger data sets?*

   The predictive accuracy of the GP always requires training data in the desired predictive region (which is true for any System Identification method). While the algorithm incorporated online updates, the kernel hyper-parameters and initial training are crucial for the success of the algorithm and therefore the offline training phase and initial validation should be emphasized.

   The GP model does not require much prior knowledge to implement to a variety of test cases, as was shown in Chapter 6 where the same Squared Exponential (SE) kernel was successfully applied to a variety of test cases.

   The algorithm proved to be computationally demanding and the computational time of solving one iteration scaled rapidly with the number of training points. Not only that, but considering problems of higher dimensionality will naturally require more training points. While this issue was partially addressed with the use of sparse kernel approximations, the nature of solving a nonlinear optimization problem will always pose problems for implementing the algorithm to systems requiring small controller sampling time (in the order of 10^{-1} or lower).

5. *Can the final algorithm be applied to process models in both simulations and physical set-ups? What are the challenges in practical implementation?*

Chapter 6 applied the algorithm to test cases in both simulation and a physical setup. While the performance was considered successful, applying the algorithm to physical setups can be a challenge if proper regressors are not used or there is no knowledge of the underlying system.

Tuning the controller for the Temperature Control Lab (TCL) physical setup proved to be a cumbersome task because of the slow moving behaviour of the temperature system. The method of tuning a controller based on a closed-loop time response behaviour may also be impossible if the underlying system has safety concerns associated with it. In that sense, the lack of mathematical stability guarantees and the occasional failure in the nonlinear problem to converge can be an issue. However, practical test cases showed that the optimization reliably converged to a good solution if the modelled dynamics performed well in the validation phase, and that the algorithm can be applied to a variety of different systems.

7-2 Future Work

While these are the concluding remarks, there is much work that can be done in this area of research. Some possible areas of investigation are:

- Investigate the use of new kernels and the increase in flexibility of tailoring a more structured kernel function to specific applications as a means of incorporating prior knowledge.
- Incorporate kernel hyper-parameter updates in an online fashion.
- Use probabilistic chance constraints in the Stochastic MPC problem.
- Investigate stability guarantees through the use of Lyapunov functions.
- Experiment with different GP approximations to tackle the problem of scalability.

These improvements were out of the scope of this thesis, but are interesting research areas which have had some attention in recent years. The rapid growth of this research field and demand for improvements in performance and mathematical guarantees will no doubt raise many more questions and considerations in the near future.

Bibliography

[1] W. Favoreel, B. D. Moor, and M. Gevers, "Spc: Subspace predictive control," *IFAC Proceedings Volumes*, vol. 32, no. 2, pp. 4004–4009, 1999. 14th IFAC World Congress 1999, Beijing, Chia, 5-9 July.

[2] O. P. Ogunmolu, X. Gu, S. B. Jiang, and N. R. Gans, "Nonlinear systems identification using deep dynamic neural networks," *CoRR*, vol. abs/1610.01439, 2016.

[3] P. Angelo and A. Drummond, "A survey of random forest based methods for intrusion detection systems," *ACM Computing Surveys*, vol. 51, 05 2018.

[4] P. Drezet and R. Harrison, "Support vector machines for system identification," pp. 688 – 692 vol.1, 10 1998.

[5] S. Särkkä, *The Use of Gaussian Processes in System Identification*, pp. 1–10. London: Springer London, 2019.

[6] C. A. Micchelli, Y. Xu, and H. Zhang, "Universal kernels," *Journal of Machine Learning Research*, vol. 7, no. 95, pp. 2651–2667, 2006.

[7] "Data-driven model predictive control using random forests for building energy optimization and climate control," *Applied Energy*, vol. 226, pp. 1252–1272, 2018.

[8] "Neural networks in system identification," *IFAC Proceedings Volumes*, vol. 27, no. 8, pp. 359–382, 1994. IFAC Symposium on System Identification (SYSID'94), Copenhagen, Denmark, 4-6 July.

[9] R. Winqvist, A. Venkitaraman, and B. Wahlberg, "On training and evaluation of neural network approaches for model predictive control," 2020.

[10] D.-H. Jung, H.-J. Kim, J. Y. Kim, T. S. Lee, and S. H. Park, "Model predictive control via output feedback neural network for improved multi-window greenhouse ventilation control," *Sensors*, vol. 20, no. 6, 2020.

[11] C. E. Rasmussen, *Evaluation of Gaussian Processes and Other Methods for Non-Linear Regression*. PhD thesis, CAN, 1997. AAINQ28300.

[12] J. Kocijan and R. Murray-Smith, "Nonlinear predictive control with a gaussian process model.," pp. 185–200, 01 2003.

[13] L. Hewing and M. Zeilinger, "Cautious model predictive control using gaussian process regression," *IEEE Transactions on Control Systems Technology*, vol. PP, 05 2017.

[14] Y. Wang, C. Ocampo-Martinez, and V. Puig, "Stochastic model predictive control based on gaussian processes applied to drinking water networks," *IET Control Theory and Applications*, vol. 10, pp. 947 – 955, 05 2016.

[15] B. van Niekerk, A. Damianou, and B. Rosman, "Online constrained model-based reinforcement learning," 04 2020.

[16] J. Umlauft and S. Hirche, "Feedback linearization based on gaussian processes with event-triggered online learning," 11 2019.

[17] J. Umlauft, T. Beckers, A. Capone, A. Lederer, and S. Hirche, "Smart forgetting for safe online learning with gaussian processes," 2020.

[18] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[19] M. Verhaegen and V. Verdult, *Filtering and System Identification: A Least Squares Approach*. 01 2007.

[20] V. Vovk, *Kernel Ridge Regression*, pp. 105–116. 10 2013.

[21] J. Kocijan, A. Girard, B. Banko, and R. Murray-Smith, "Dynamic systems identification with gaussian processes," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 11, 12 2005.

[22] G. E. Hinton and R. Neal, "Bayesian learning for neural networks," 1995.

[23] S. Faul, G. Gregorcic, G. Boylan, W. Marnane, G. Lightbody, and S. Connolly, "Gaussian process modeling of eeg for the detection of neonatal seizures," *IEEE transactions on biomedical engineering*, vol. 54, pp. 2151–62, 01 2008.

[24] M. K. Titsias, M. Rattray, and N. D. Lawrence, *Markov chain Monte Carlo algorithms for Gaussian processes*, p. 295–316. Cambridge University Press, 2011.

[25] M. Titsias, "Variational learning of inducing variables in sparse gaussian processes," *Journal of Machine Learning Research - Proceedings Track*, vol. 5, pp. 567–574, 04 2009.

[26] A. Girard and R. Murray-Smith, "Gaussian processes: Prediction at a noisy input and application to iterative multiple-step ahead forecasting of time-series," *Lecture Notes in Computer Science*, vol. 3355, pp. 158–184, 01 2003.

[27] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, p. 408–423, Feb 2015.

[28] L. Hewing, A. Liniger, and M. Zeilinger, "Cautious nmpc with gaussian process dynamics for autonomous miniature race cars," pp. 1341–1348, 06 2018.

[29] A. Girard, *Approximate Methods for Propagation of Uncertainty with Gaussian Process Models*. University of Glasgow, 2004.

[30] P. Groot, P. J. Lucas, and P. Bosch, "Multiple-step time series forecasting with sparse gaussian processes," *Journal of Vegetation Science - J VEG SCI*, 01 2011.

[31] T. Gutjahr, "Sparse gaussian processes with uncertain inputs for multi-step ahead prediction," *IFAC Proceedings Volumes*, vol. 45, pp. 107–112, 07 2012.

[32] A. Mchutchon and C. Rasmussen, "Gaussian process training with input noise," *NIPS*, 01 2011.

[33] A. McHutchon, "Differentiating gaussian processes," 04 2013.

[34] H. Bijl, T. Schön, J. W. Wingerden, and M. Verhaegen, "System identification through online sparse gaussian process regression with input noise," *IFAC Journal of Systems and Control*, vol. 2, pp. 1–11, 12 2017.

[35] H. Bijl, J. W. Wingerden, T. Schön, and M. Verhaegen, "Online sparse gaussian process regression using fitc and pitc approximations," *IFAC-PapersOnLine*, vol. 48, pp. 703–708, 12 2015.

[36] M. Maiworm, D. Limon, and R. Findeisen, "Online gaussian process learning-based model predictive control with stability guarantees," 11 2019.

[37] T. D. Bui, C. Nguyen, and R. E. Turner, "Streaming sparse gaussian process approximations," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[38] H. Liu, Y. Ong, X. Shen, and J. Cai, "When gaussian process meets big data: A review of scalable gps," 07 2018.

[39] A. Smola and P. Bartlett, "Sparse greedy gaussian process regression," *NIPS*, vol. 13, 02 2001.

[40] M. Seeger, C. Williams, and N. Lawrence, "Fast forward selection to speed up sparse gaussian process regression," 06 2014.

[41] J. Quiñonero-Candela and C. E. Rasmussen, "A unifying view of sparse approximate gaussian process regression," *Journal of Machine Learning Research*, vol. 6, no. 65, pp. 1939–1959, 2005.

[42] V. Lalchand and A. C. Faul, "A fast and greedy subset-of-data (sod) scheme for sparsification in gaussian processes," 2020.

[43] E. Snelson and Z. Ghahramani, "Sparse gaussian process using pseudo-inputs," *Advances in Neural Information Processing Systems*, vol. 18, 01 2006.

[44] C. Williams and M. Seeger, "Using the nyström method to speed up kernel machines," in *Advances in Neural Information Processing Systems* (T. Leen, T. Dietterich, and V. Tresp, eds.), vol. 13, MIT Press, 2001.

[45] S. Keerthi and W. Chu, "A matching pursuit approach to sparse gaussian process regression," vol. 18, 01 2005.

[46] M. Seeger, C. Williams, and N. Lawrence, "Fast forward selection to speed up sparse gaussian process regression," 06 2014.

[47] N. Lawrence, M. Seeger, and R. Herbrich, "Fast sparse gaussian process methods: The informative vector machine," *Adv NIPS*, vol. 15, pp. 609–616, 01 2002.

[48] J. Schreiter, D. Nguyen-Tuong, and M. Toussaint, "Efficient sparsification for gaussian process regression," *Neurocomputing*, 03 2016.

[49] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Advances in Neural Information Processing Systems* (J. Platt, D. Koller, Y. Singer, and S. Roweis, eds.), vol. 20, Curran Associates, Inc., 2008.

[50] M. Lázaro-Gredilla, J. Quiñnero-Candela, C. E. Rasmussen, and A. R. Figueiras-Vidal, "Sparse spectrum gaussian process regression," *Journal of Machine Learning Research*, vol. 11, no. 63, pp. 1865–1881, 2010.

[51] C. E. Rasmussen and H. Nickisch, "Gaussian processes for machine learning (gpml) toolbox," *J. Mach. Learn. Res.*, vol. 11, p. 3011–3015, Dec. 2010.

[52] T. Yang, Y.-f. Li, M. Mahdavi, R. Jin, and Z.-H. Zhou, "Nyström method vs random fourier features: A theoretical and empirical comparison," vol. 25, 2012.

[53] R. B. Gramacy and H. K. H. Lee, "Cases for the nugget in modeling computer experiments," 2010.

[54] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[55] E. Bradford, L. Imsland, D. Zhang, and E. del Rio-Chanona, "Stochastic data-driven model predictive control using gaussian processes," 08 2019.

[56] B. Roffel and B. H. Betlem, *Internal Model Control*, pp. 161–169. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[57] A. J. Krener, *Feedback Linearization*, pp. 66–98. New York, NY: Springer New York, 1999.

[58] J. Umlauft, T. Beckers, M. Kimmel, and S. Hirche, "Feedback linearization using gaussian processes," pp. 5249–5255, 12 2017.

[59] G. Gregorcic and G. Lightbody, "Gaussian process for internal model control," *International Journal of Systems Science - IJSySc*, vol. 43, 01 2011.

[60] L. Grne and J. Pannek, *Nonlinear Model Predictive Control: Theory and Algorithms*. Springer Publishing Company, Incorporated, 2013.

[61] A. Mills, *Basic Heat and Mass Transfer*. Prentice Hall, 1999.

Glossary

List of Acronyms

GP	Gaussian Process
GPR	Gaussian Process Regression
MPC	Model Predictive Control
ARD	Automatic Relevance Determination
SE	Squared Exponential
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
SMSE	Standardized Mean Squared Error
LDE	Log Density Error
MSLL	Mean Standardized Log Loss
AIC	Akieke's Information Criterion
NARX	Nonlinear AutoRegressive eXogenous
NAR	Nonlinear AutoRegressive
NFIR	Nonliner Finite Impulse Response
EEG	EletroEncephaloGram
SSM	State Space Model
MCMC	Markov Chain Monte Carlo
FITC	Fully Independent Training Conditionals
SS	Sparse Spectrum
SoR	Subset of Regressors
SPGP	Sparse Pseudo-input Gaussian Process
DTC	Deterministic Training Conditionals
PITC	Partially Independent Training Conditionals

NIGP	Noisy Input Gaussian Process
SISO	Single Input Single Output
MIMO	Multiple Input Multiple Output
IMC	Internal Model Control
TCL	Temperature Control Lab
RWP	Reaction Wheel Pendulum