# GAUSSIAN PROCESSES FOR ONLINE LEARNING-BASED MODEL PREDICTIVE CONTROL

SUMANTA GHOSH

(2019EEY7566)


DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY DELHI

JUNE 2022

# Gaussian Processes for Online Learning-Based Model Predictive Control

*Thesis submitted by*

**SUMANTA GHOSH**

**(2019EEY7566)**

*under the guidance of*

**Dr.Shubhendu Bhasin**

*partial fulfilment of the requirements*
*for the award of the degree of*

**Master of Science (Research)**


**Department of Electrical Engineering**
**INDIAN INSTITUTE OF TECHNOLOGY DELHI**

**June 2022**

# Certificate

This is to certify that the thesis entitled **"Gaussian Processes for Online Learning Based Model Predictive Control"**, submitted by **SUMANTA GHOSH** to the Indian Institute of Technology Delhi, for the award of the degree of **Master of Science (Research)** in Electrical Engineering, is a record of the original, bona fide research work carried out by him under my supervision and guidance. The thesis has reached the standards fulfilling the requirements of the regulations related to the award of the degree.

The results contained in this thesis have not been submitted in part or in full to any other University or Institute for the award of any degree or diploma to the best of our knowledge.

**Prof. Shubhendu Bhasin**
Department of Electrical Engineering,
Indian Institute of Technology Delhi.
Date: May 2022.

# *Acknowledgements*

I would like to thank my thesis supervisor Prof. Shubhendu Bhasin for his guidance, advice, support, encouragement, and patience during my MS(R) studies. I would highly appreciate the overall guidance, constructive suggestions and recommendations provided my SRC members Prof. Indra Narayan Kar, Prof. Deepak Patil, Prof. Sivananthan Sampath for different aspects of my MS(R) journey. I am always grateful to the other faculty and staff members of the IIT Delhi, Control and Automation group for their help and support.

Sumanta Ghosh

# *Abstract*

Optimal control is a well established field that focuses on finding a control law that minimizes a certain performance measure. This kind of control methodology is also known as model driven approach as they relied heavily on the availability of a suitable system model. Finding a suitable model of the system is not always possible which limits its application. Learning-based methods are useful when very little information is available. These methods learn the unknown system model from the available data, opening up a large potential application of learning-based approaches.

Model predictive control (MPC) is a very popular approach to solve the optimal control problem. It can handle constraints on the system states and the inputs ensuring safety of the system. The performance of MPC depends on the model of the system, as it uses the model to simulate the future states. Thus, having an appropriate system model is very critical for the performance of the control law. Gaussian processes (GPs) are very useful for modelling the system dynamics from data as it quantify the uncertainty within the learnt model by providing a probabilistic interpretation of it. The uncertainty in the model is then utilised to predict the future states in order to improve the performance.

Learning system dynamics in an online setting is very significant since it aids in dealing with changing circumstances. Due to the limitations with streaming data, GPs are not a popular choice for online learning. This dissertation proposes a learning based MPC approach that overcomes the barrier of online learning with Gaussian processes by using kernel interpolation scheme and achieves better performance than the existing offline learning based MPC.

# Contents

# List of Figures

# Abbreviations

| | |
|---|---|
| **MPC** | **M**odel **P**redictive **C**ontrol |
| **RL** | **R**einforcement **L**earning |
| **ML** | **M**aximum **L**ikelihood |
| **GP** | **G**aussian **P**rocess |
| **GPMPC** | **G**aussian **P**rocess **M**odel **P**redictive **C**ontrol |
| **NN** | **N**eural **N**etwork |
| **MM** | **M**oment **M**atching |
| **ELF** | **E**xtended **K**alman **F**ilter |

# Chapter 1

# Introduction

Classical optimal control techniques have been successfully applied in several fields, including process control [1], aerospace [2], robotics [3], economics, and finance [4]. The performance of these techniques depends on the availability of good system model. Recent developments in computing, communication, and sensing technologies have opened new potential applications of optimal control. Examples include autonomous vehicles, physical human-robot interaction, and advanced process control. The system dynamics corresponding to these applications is often unknown or partially known with uncertainties. Uncertainties may arise from various sources. For example, the mass distribution of a payload carried by an unmanned aerial vehicle (UAV) may not be known a priori or the real-time weather may change suddenly (e.g., snow, rain, and hail). Deriving an accurate analytical model for these systems is often very difficult, sometimes impossible, depending on the complexity of the system. As a result, the performance of these techniques becomes unsatisfactory, motivating the necessity of an alternative approach to tackle the problem of model-driven approach.

Machine learning techniques have become very popular thanks to the availability of good quality data. Applications of these techniques are widespread, including image

classification [5], computer vision [6], natural language processing [7], time series analysis [8], autonomous driving [9], and robotics [10]. The supervised learning paradigm is useful for obtaining a model from labeled data with little or no prior information about the model. Popular supervised learning models include Linear regression, Neural networks (NN), and Gaussian processes (GPs). Another popular machine learning paradigm is reinforcement learning, where an agent learns to map a sequence of observations to actions to achieve a particular goal by maximizing a numerical reward function over time.

## 1.1 Preliminaries

### 1.1.1 Dynamical Systems

A very important component of control theory is dynamical systems; systems that evolve over time. Discrete-time dynamical systems where the time variable is treated as discrete are considered in this work. Throughout this thesis, we consider a deterministic discrete-time dynamical system where the behavior of the system at every time step $k$ can be completely described by the state $\mathbf{x}_k$ at that time. Control input $\mathbf{u}_k$ is applied at every time step to drive the system to the desired state.

$$\mathbf{x}_{k+1} = \boldsymbol{f}(\mathbf{x}_k, \mathbf{u}_k) \tag{1.1}$$

Here, $\boldsymbol{f}$ is the system's transition function, commonly referred to as the system evaluation function. In our scenario, $\boldsymbol{f}$ is either unknown or only partially known. In the control literature, the availability of a nominal model derived from the first principles is a common assumption. For this case, we can divide $\boldsymbol{f}$ into two parts: $\boldsymbol{h}$, which consists of the known component or the nominal model and $\boldsymbol{g}$, the unknown

or uncertain part which represents the initially unmodeled dynamics of the system. These kinds of systems can be represented as,

$$\mathbf{x}_{k+1} = \boldsymbol{h}(\mathbf{x}_k, \mathbf{u}_k) + \boldsymbol{g}(\mathbf{x}_k, \mathbf{u}_k) \tag{1.2}$$

### 1.1.2 Optimal Control

Optimal control deals with the problem of finding a control law for a dynamical system that achieves a certain optimality criterion over a period of time. The optimality criterion is usually defined with respect to an objective function, a function of state and control variables. The goal is to find a control law that achieves a minimum value of the objective function over a specified amount of time maintaining the system constraints.

We consider discrete-time dynamical system defined in Eq. (1.1) with states $\mathbf{x}_k \in \mathcal{X} \subset \mathbb{R}^{n_x}$ and control input $\mathbf{u}_k \in \mathcal{U} \subset \mathbb{R}^{n_u}$. We assume that the system obeys the Markov property; that is, given the current state $\mathbf{x}_k$, and the current control input, $\mathbf{u}_k$, the next state $\mathbf{x}_{k+1}$ is completely specified by the state transition function ($\boldsymbol{f}$). The system dynamics is considered as known and we have direct access to the noisy measurements of the system states. The objective function $J$ is defined as the cumulative sum of stage costs, $l(\mathbf{x}_k, \mathbf{u}_k)$.

$$J = \sum_{k=0}^{N-1} l(\mathbf{x}_k, \mathbf{u}_k) \tag{1.3}$$

The aim is to find a control sequence $(\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_{N-1})$ and corresponding state sequence that $(\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_N)$ that minimizes the objective function. In the next

section, we will discuss a class of control algorithms that repeatedly solves a finite-horizon constrained optimal control problem to achieve the desired system performance.

### 1.1.3 Model Predictive Control

Model Predictive Control (MPC), also known as receding horizon control, is one of the most successful and popular advanced control methods. It has a wide range of applications, including hybrid electric vehicles [11], smart buildings [12], process control [13, 14], control of electrical drives [15, 16, 17], flight control [18], etc. The reason behind the popularity of MPC is its constraint handling capabilities. In real world systems, the control variables are constrained due to the physical limitations of actuators. Also constraints can be imposed on the states owing to the safety reasons. If the system variables satisfies the specified system constraints for all the time, the system is said to be safe. Suppose for a system with transition function $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k)$, the system constraint set is denoted by $\mathcal{X}$ for system variable $\mathbf{x}$ and the control constraint set is denoted by $\mathcal{U}$ for control $\mathbf{u}$ .Then the system can be defined as safe if it satisfies the following condition,

$$\forall k \in \mathbb{N}: \quad f(\mathbf{x}_k, \mathbf{u}_k) \in \mathcal{X}, \quad \mathbf{u}_k \in \mathcal{U}$$

MPC utilizes the system model for iterative prediction of its future states to compute the objective function. Thus, the performance of a MPC controller depends on availability of an accurate model of the system, a suitable cost function, and constraints formulation. At each instant of time an optimal control problem, Eq.(1.4)-Eq.(1.8) is solved over a finite prediction horizon using the current measurement of the state as the initial point. Solving the optimization problem results in an optimal sequence

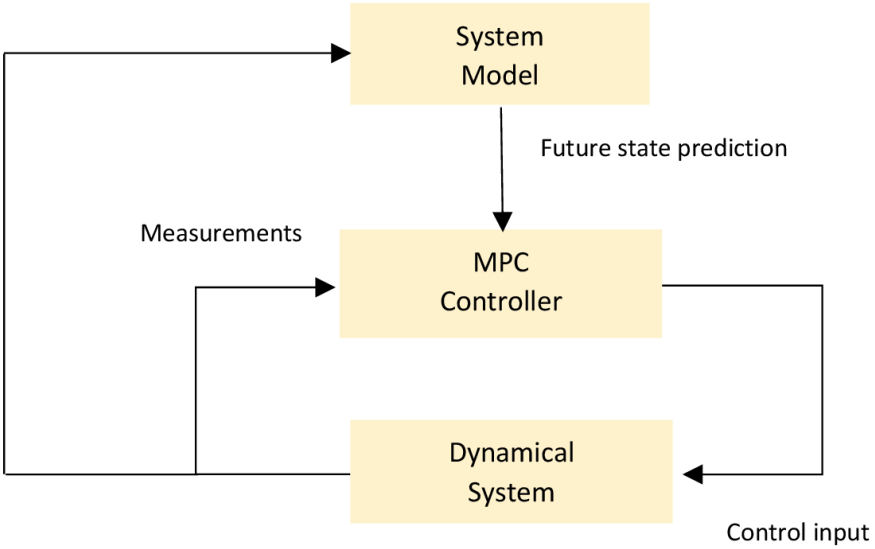


FIGURE 1.1: Schematic diagram of MPC

of controls, from which the first control is applied to the system. The dynamical system changes its state, and new state measurements are obtained. Again, the optimal control is solved at the next time instant with new initial point shifting the prediction horizon by one step. This process is repeated continuously as time progress. A schematic diagram of MPC is shown in Figure 1.1.

The MPC online optimization problem at time instant $k$ is defined in Eq.(1.4) - Eq.(1.8). As discussed before, some of the essential parts of the optimization problem are the model of the system Eq.(1.5), state constraints Eq.(1.6), input constraints Eq.(1.7). The objective function which is the total accumulation of the stage costs ($l_i$) over time plus the terminal cost ($l_f$) is represented in Eq.(1.4). The initial point

of the optimization problem is updated with new state measurements in Eq.(1.8).

$$\min_{\mathbf{u}_0,\mathbf{u}_1,..,\mathbf{u}_{N-1}} \left( l_f(\mathbf{x}_N) + \sum_{i=0}^{N-1} l_i(\mathbf{x}_i, \mathbf{u}_i) \right) \tag{1.4}$$

$$\text{s.t.} \quad \mathbf{x}_{i+1} = \boldsymbol{f}(\mathbf{x}_i, \mathbf{u}_i) \tag{1.5}$$

$$\mathbf{x}_{i+1} \in \mathcal{X} \tag{1.6}$$

$$\mathbf{u}_i \in \mathcal{U} \tag{1.7}$$

$$\mathbf{x}_0 = \mathbf{x}(k) \tag{1.8}$$

The theory of model predictive control has been well developed, mostly for systems with linear dynamics. As the optimization problem is solved on-line, problems like stability and recursive feasibility of the optimization problem may need to be addresses. The issue of stability and recursive feasibility of the on-line optimization problem is broadly discussed in [19, 20, 21, 22]. The theory related to robust model predictive control can be found in [23, 24].

### 1.1.4 Reinforcement Learning

Reinforcement learning [25] is a part of the machine learning paradigm that focuses on learning to find the control signal that minimizes a mathematical measure that expresses a long-term objective. The idea of reinforcement learning (RL) is closely related to optimal control and adaptive control. In the previous section, we discussed optimal control problems in which the goal is to design controllers to minimize a system-dependent objective function over time. Determining optimal control policies for nonlinear systems often requires offline computations and knowledge of the system dynamics. In contrast, adaptive controllers [26] learn online to control unknown systems using real-time data measured along the trajectories. Usually,

adaptive controllers are not designed to be optimal in the sense of minimizing a user defined performance function. In reinforcement learning, ideas from adaptive control and optimal control are combined. In the context of control systems, RL refers to a class of methods that enable the design of adaptive controllers with approximate solutions to the optimal control problem [27].

#### 1.1.4.1 Elements of Reinforcement Learning

Reinforcement learning is a class of algorithms that solves a problem by learning from interaction. The learner or the decision maker is called an agent. It interacts with the environment- everything outside the agent. The agent performs actions; the environment responds to these actions by presenting new situations to the agents. The environment also emits rewards, special numerical values that the agent tries to maximize over time. A typical setting for the agent-environment interaction for the reinforcement learning task is shown in Figure 1.2. In control theory, the agent and the environment are known as the controller and the plant, respectively. The reward in reinforcement learning plays a role similar to the stage cost function in optimal control, but in an opposite sense. Throughout this work, we consider control system-related terminologies.

Reinforcement learning algorithms can be divided into two groups, model-free reinforcement learning algorithms (MFRL) and model-based reinforcement learning algorithms (MBRL). The model-free reinforcement learning algorithms [28] focuses on learning the value functions from interaction with the system. This type of algorithm has achieved a large amount of success mostly in computer games and simulated worlds [29]. The reason behind the limited application of these algorithms is data inefficiency. On the other hand, model-based reinforcement algorithms [30] learn a predictive model of the system and then use the learned model for decision

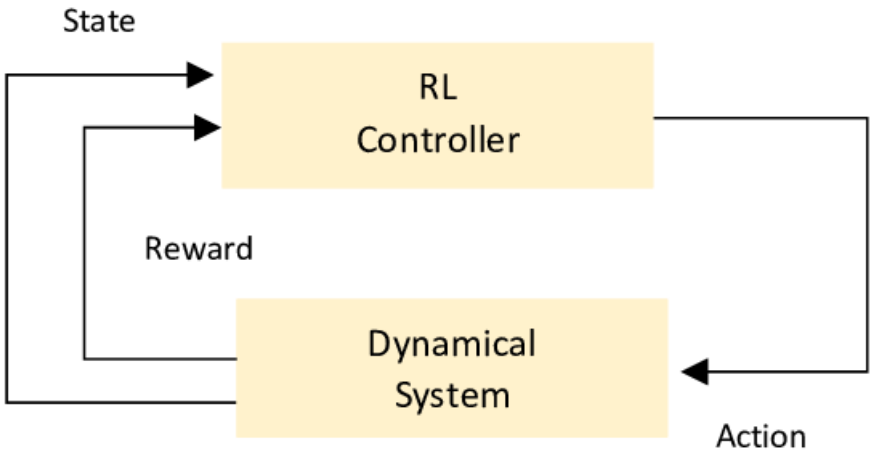


FIGURE 1.2: The basic reinforcement learning scenario

making. This kind of algorithms are highly data-efficient and suitable for real-world applications. In addition, the learned model is reward-independent and can be generalized to new tasks in the same environment. This enables the application of MBRL algorithms to a wide variety of fields. In the next section, we will discuss briefly the model-based reinforcement algorithms.

#### 1.1.4.2 Model Based Reinforcement Learning

In model-based reinforcement learning, the dynamics of the system is learned using the available data (observations). Learning the dynamics is a task of fitting an approximation $\tilde{f}$ of the true transition function, given the measurements $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{u}_i), \mathbf{x}_{i+1}\}_{i=1}^{n}$ of the real world. The learned dynamics model $\tilde{f}$ is then used to predict the distribution over the state trajectories resulting from applying a sequence of control inputs. We can compute the total expected cost over a state

trajectories and this gives different expected long-term cost for different candidate control sequences. The optimal control sequence gives the solution to the reinforcement learning problem. Mathematically, this can be formulated as follows.

$$\mathbf{u}^{\star} = \arg\min_{\mathbf{u}} \sum_{k=0}^{N} \mathbb{E}_{\mathbf{x}_k,\mathbf{u}_k} \left[ l(\mathbf{x}_k, \mathbf{u}_k) \right] \tag{1.9}$$

Where, $\mathbb{E}_{\mathbf{x},\mathbf{u}} \left[ l(\mathbf{x}, \mathbf{u}) \right]$ represents the expectation of $l(\mathbf{x}, \mathbf{u})$ with respect to random variables $\mathbf{x}$ and $\mathbf{u}$. p($\mathbf{x}$,$\mathbf{u}$) denotes the joint distribution of $\mathbf{x}$ and $\mathbf{u}$.

$$\mathbb{E}_{\mathbf{x},\mathbf{u}} \left[ l(\mathbf{x}, \mathbf{u}) \right] = \int_{\mathbf{x}\in\mathcal{X}} \int_{\mathbf{u}\in\mathcal{U}} l(\mathbf{x}, \mathbf{u}) p(\mathbf{x}, \mathbf{u}) d\mathbf{x} d\mathbf{u}$$

#### 1.1.4.3 Uncertainty in Model Learning

In model-based reinforcement learning, the learned model plays a crucial role in achieving the desired performance of the system, as it is used to simulate the system internally to predict the long-term behavior of the system without directly interacting with it. Based on these simulations, the optimal policy is found. Use of a learned model to speculate on the future behavior of the system often results in poor performance as any small error in it can accumulate while predicting the long-term behavior.

State prediction errors can be mitigated largely by properly incorporating uncertainty into the learned dynamical model. There are two distinct classes of uncertainty [31] associated with a learned model. One is aleatoric uncertainty, arises due to inherent stochasticity of the system. For example, the model may be less certain about the data because of the high observational noise. The second type is epistemic uncertainty, which emphasizes our confidence in the learned model under the assumption of true data. This kind of uncertainty arises due to limited amount of

data or the use of a less expressive model class. Isolating epistemic uncertainty is important in model-based reinforcement learning as it helps in better understanding of the learned model.

One way to reduce epistemic uncertainty is the use of a large amount of data. In the limit of infinite data, epistemic uncertainty vanishes, but for data sets of finite size, this uncertainty remains and induces error while predicting the state transitions. Another way to reduce the epistemic uncertainty is the use of more expressive model class. Artificial neural networks (NNs) are a very popular choice for learning an unknown function from data. NNs are scalable to large data-sets, have constant inference time, and most importantly have the potential to represent more complex models. However, NNs suffer from over-fitting on small data sets, resulting in poor state predictions far in the future at the early stages of learning. Bayesian non-parametric Gaussian processes are known to work well in low data regimes [32]. It gives a distribution over the function that quantifies the uncertainty in the prediction. This measure of uncertainty is used further to make better prediction of future states. That is why Gaussian processes are more suitable for model-based reinforcement learning. We will discuss Gaussian processes in great detail in Chapter 2.

### 1.1.5 Learning Based MPC

Previously we have seen two different approaches to solve the optimal control problem. One uses a system model to analyze future behavior and the other learns the model from the available data. Both have their own limitations. MPC requires an exact system model, well defined objective function, and constraints. For this reason, designing an MPC controller requires expert knowledge of the system. RL uses

minimal domain knowledge and learns from interactions with the world. MPC solves the constraint optimization problem online, requiring more computational resources but it guarantees the constraint satisfaction making it highly useful in safety-critical task. On the other hand, RL does not guarantee constraint satisfaction.

Learning-based model predictive control (LMPC) combines these two approaches. It uses a statistical model to learn the unknown dynamics from the measurements, thereby reducing the dependency on expert knowledge. Then it uses the learned model to plan the future behaviour to evaluate the objective function and solves a nonlinear constraint optimization problem at each time instant. So, LMPC performs two important steps, $(i)$ learning the unknown system dynamics and planning through future states and $(ii)$ determining the control by solving MPC optimization problem. The first step may be performed offline, where the model of the system is learned from previously collected data. The work in this thesis is founded on online learning since it gives flexibility with changing system conditions. We also consider that the nominal model of the system derived from the first principles is available. A schematic diagram of LMPC is shown in Figure 1.3.

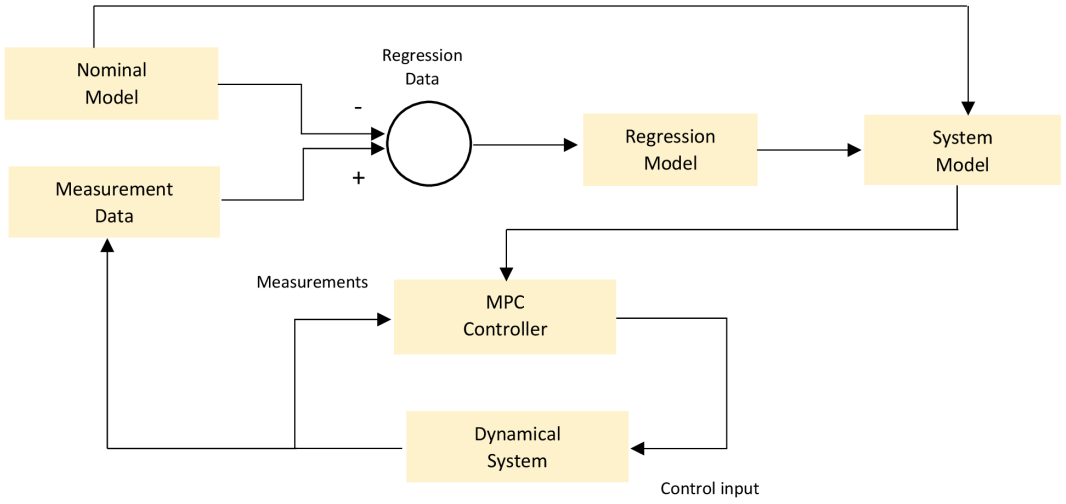


FIGURE 1.3: The schematic diagram of learning-based MPC

## 1.2 Problem Statement

We consider a deterministic discrete-time dynamical system given by

$$\mathbf{x}_{k+1} = \boldsymbol{f}(\mathbf{x}_k, \mathbf{u}_k) = \boldsymbol{h}(\mathbf{x}_k, \mathbf{u}_k) + \boldsymbol{g}(\mathbf{x}_k, \mathbf{u}_k) \tag{1.10}$$

where $\mathbf{x}_k \in \mathcal{X} \subset \mathbb{R}^{n_x}$ and $\mathbf{u}_k \in \mathcal{U} \subset \mathbb{R}^{n_u}$. We assume that the system dynamics is partially known with the nominal model $\boldsymbol{h}$ and, the unknown or uncertain part $\boldsymbol{g}$. We use a probabilistic model, $\hat{\boldsymbol{g}}$ to learn the unknown system dynamics from the available data. The use of a probabilistic model gives us the flexibility to use the model uncertainty (epistemic uncertainty) in our state prediction. This results in a stochastic state distribution. We formulate the constrained optimal control problem as a stochastic MPC problem as given by Eq. (1.11)-(1.15). The constraints are formulated as chance constraints in Eq. (1.13) and Eq. (1.14) with the probability of constraint satisfaction $p_x$ and $p_u$ for states and control, respectively. These values are user-defined and reflects the confidence on the model. At each time instant, the LMPC solves the following optimization problem.

$$\min_{\mathbf{u}_0, \mathbf{u}_1, .., \mathbf{u}_{N-1}} \mathbb{E}\left( l_f(\mathbf{x}_N) + \sum_{i=0}^{N-1} l_i(\mathbf{x}_i, \mathbf{u}_i) \right) \tag{1.11}$$

$$\text{s.t.} \quad \mathbf{x}_{i+1} = \boldsymbol{h}(\mathbf{x}_i, \mathbf{u}_i) + \hat{\boldsymbol{g}}(\mathbf{x}_i, \mathbf{u}_i) \tag{1.12}$$

$$\mathbf{Pr}(\mathbf{x}_{i+1} \in \mathcal{X}_{i+1}) \geq p_x \tag{1.13}$$

$$\mathbf{Pr}(\mathbf{u}_i \in \mathcal{U}_i) \geq p_u \tag{1.14}$$

$$\mathbf{x}_0 = \mathbf{x}(k) \tag{1.15}$$

## 1.3 Literature Survey

System identification based on learning a dynamical model from data has gained significant attention in recent years [33, 34, 35]. Learning dynamical system has reduced the dependency on idealized assumptions about the system [36]. For example, [37] uses a feedforward neural network for dynamical system identification and control. Bayesian non-parametric models have become very popular in system identification due to their learning ability with a smaller data set and quantification of uncertainty about the model [38, 39]. Planning through the learned system dynamics becomes challenging with imperfect state information. Efficient methods to evaluate GPs for Gaussian input have been developed in [40, 41, 42, 43]. A framework for learning and control using Gaussian process-based modeling and uncertainty propagation is given in [44, 45].

Efficient Gaussian process model based predictive control was first presented in [46]. Learning time-varying disturbances using GP models to improve MPC controller performance is shown in [47]. In [48] a learning-based nonlinear model predictive control algorithm is proposed to achieve tracking performance in challenging off-road terrain. This highlights the expressiveness of the Gaussian process regression model for learning complex dynamics from data. [49] developed a data-efficient reinforcement learning algorithm with model predictive control policy. This achieves better data efficiency in performing certain benchmark tasks compared to existing reinforcement learning algorithms. [50, 51] presented a learning based model predictive control approach that integrates a nominal system with additive nonlinear part modeled as GP and developed a probabilistic reachable set [52] based chance constraint formulation approach. An application of the approach presented in [51] for data driven control of quad-rotors is demonstrated in [53]. A data driven model

predictive control for trajectory tracking with robotic arm was shown [54]. This paper developed an approach that combines a GP based model trained offline with an additive disturbance model estimated online by an extended Kalman filter (EKF).

Application of Gaussian processes in online learning scenario is limited due to its higher computational cost for prediction and hyperparameter learning. The sparse approximation technique mentioned in [50, 51] uses the inducing points method [55] that summarizes the whole data set into a subset of constant size. This approximation achieves better scalability of model to large data sets but with reduced performance. In addition, online learning typically uses measurements that are available sequentially. To update the GP with new measurements one has to train it again on the whole data set augmented with the new data points. That makes application of GP in online setting more challenging. The approach presented in [56] uses a fixed-size data set to train the model and discards the oldest data point when new measurements are available. This gives a local approximation of the function, forgetting the previous experience.

## 1.4 Contribution

In learning based MPC the model learning is usually performed offline using the previously collected data. Then the learned model is used for predicting future states. Difficulty arises when LMPC is used in online learning scenario when measurements are available sequentially. Existing approaches for online LMPC use a fixed size data set to overcome the difficulty of online learning and develop rule to update data set with new measurements. Use of fixed size data set limits the expressiveness of the model class. The proposed approach does not consider any constraint on the size of

the data set and learns an unknown model online. Our contribution lies in developing an approach that combines kernel interpolation approach for online Gaussian processes with model predictive control to achieve better performance in an online learning scenario.

## 1.5 Thesis Organization

The thesis is organised into five chapters. Each chapter is composed in a self-consistent manner. The summary of the work presented in each chapter is briefly outlined as follows:

**Chapter 1:** This is an introductory chapter that elaborates the motivation of the research, problem formulation, literature review, contribution, and an outline of the thesis.

**Chapter 2:** In this chapter we introduce Gaussian processes for system identification. This contains an overview of Gaussian process regression, hyperparameter tuning, prediction with uncertain input (Gaussian), Gaussian process state-space model (GPSS), and online structured kernel interpolation for online Gaussian process regression (WISKI).

**Chapter 3:** In this chapter we discuss our approach to online learning based model predictive control. We discuss model learning and uncertainty propagation, selection of cost function and chance constraint formulation.

**Chapter 4:** Simulation results are presented in this chapter with a detailed analysis.

**Chapter 5:** This chapter presents the conclusion of the thesis and a brief discussion of the future work.

# Chapter 2

# Gaussian Processes for System Identification

## 2.1 Introduction

Learning from data has become a very powerful tool to understand system with complex dynamics. Having a suitable dynamical model of the system is very crucial for designing controllers, as the model is used to make predictions about the evolution of the system under a control effort. Data-driven approaches are often termed "black box modelling" as they allow us to model complex dynamics without understanding the underlying mathematics. In the data-driven approach, a class of models is typically chosen in advance, and the observations are then utilised to select a model from that particular model class. This procedure is referred to as model selection in machine learning paradigm [57]. For nonlinear systems, selecting a suitable class of models to fit the data is usually very challenging. If the model

class is too simplistic, then the resulting model will be a poor approximate (under-fit the data) of the real system. Use of a more expressive model class on a small data set suffers from the over-fitting problem. The approximate model of the system performs well on the previous observations (training data) but generalizes poorly to unseen data points.

Bayesian non-parametric models are very useful for learning complex dynamical systems from data. These are a class of models that are parameterised directly by observations. As a result, they can adapt their model complexities to the data. A very popular Bayesian non-parametric model class is Gaussian processes (GPs). GPs give a probabilistic interpretation of the data that allows us to quantify the uncertainty within the model. This is very important for learning with limited amounts of data, as we can always find a large number of models within a model class that can possibly fit the data. This type of model uncertainty is known as epistemic uncertainty. Gaussian processes are capable of capturing epistemic uncertainty and reflects its confidence on the model by providing a probabilistic interpretation of data. Deterministic models offer model confidence even when there are few or no data points since they only provide one explanation for the data. This will have an adverse effect on systems with safety-critical application. In the following section, we will discuss Gaussian processes in great detail.

## 2.2 Gaussian Processes

Gaussian process(GP)[58] is a generalization of the Gaussian probability distribution over functions. It is formally defined as a collection of random variables, any finite number of which have a joint Gaussian distribution. Although a GP is of infinite dimension, we are interested in a finite subset of it. It can be completely specified by

its mean function $m(\mathbf{x})$ and its covariance function $k(\mathbf{x}, \mathbf{x}')$. For a random process $f(\mathbf{x})$, these two are defined as,

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] \tag{2.1}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}')] \tag{2.2}$$

In the previous expressions the random variables are the function values at location $\mathbf{x}$. For GPs, any finite subset of the function values is distributed jointly as Gaussian. The argument $\mathbf{x}$ of the random function plays the role of index of the random process. GPs are formally represented as,

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{2.3}$$

Usually, the mean function is defined to be zero. But one can choose other mean functions such as constant, polynomial, etc, depending on the data. Covariance function plays an important role in Gaussian process as it encodes our assumptions about the function to be learned. Notice that in Eq. (2.2), the covariance between the function values depends on the corresponding input locations, not on the actual values of the function. For GPs, the covariance function defines similarity or nearness of the function values, which means that two input points that are close to each other are likely to have similar function values. We will talk more about covariance function in context of regression problem in section 2.4.

## 2.3 Gaussian Process Regression

Supervised learning is a class of machine learning algorithms that uses labeled training data to learn an unknown mapping from feature space to target space. It can

be divided into regression and classification problems. In the regression problem, we are interested in the prediction of continuous quantities, whereas in the classification problem, the outputs are a discrete class of labels. In this literature, we mainly focus on regression problems using Gaussian process models. Here, we consider an unknown function $f(\mathbf{x})$, that maps a $D$-dimensional input $\mathbf{x}$ to a scalar function $f$. Further, we assume that we have access to noisy observations, $\{y_i\}_{i=1}^n$ of the latent function corresponding to a set of $n$ input points, $\{\mathbf{x}_i\}_{i=1}^n$. The input data points are stacked vertically to construct the input data matrix $\mathbf{X}$ of dimension $n \times D$. Similarly, we can construct the $n$-dimensional target vector, $\mathbf{y}$ and the tuple of them is formally known as training data set, $\mathcal{D} = (\mathbf{X}, \mathbf{y})$. We are interested in making an inference between the inputs and targets by finding the posterior distribution on the unknown function values given the observed data and a GP prior of the function.

We consider that the true function values are corrupted by additive independent and identically distributed zero mean Gaussian noise $\epsilon$, with noise variance $\sigma_\epsilon^2$.

$$y_i = f(\mathbf{x}_i) + \epsilon_i \tag{2.4}$$

The above assumption results in Gaussian likelihood of the data. For a data set of $n$ points, the likelihood of the data can be computed as

$$p(\mathbf{y}|f, \mathbf{X}) = \mathcal{N}(\mathbf{y}; f, \sigma_\epsilon^2 I) = \prod_{i=1}^{n} \mathcal{N}(y_i; f_i, \sigma_\epsilon^2) \tag{2.5}$$

where, $\mathcal{N}(\mu, \sigma^2)$ represents a Gaussian distribution with mean $\mu$ and variance $\sigma^2$. Finding the posterior distribution of the Gaussian Process on function values at new input points (test inputs) is our subject of interest. From the definition of GP, these function values, $\mathbf{f}^\star$ at test inputs, $\mathbf{X}^\star$ will be jointly Gaussian with the observations

$\mathbf{y}$,

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}^\star \end{bmatrix} \sim \mathcal{N}\left( \begin{bmatrix} m(\mathbf{X}) \\ m(\mathbf{X}^\star) \end{bmatrix}, \begin{bmatrix} \mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_\epsilon^2 I & \mathbf{K}(\mathbf{X},\mathbf{X}^\star) \\ \mathbf{K}(\mathbf{X}^\star,\mathbf{X}) & \mathbf{K}(\mathbf{X}^\star,\mathbf{X}^\star) \end{bmatrix} \right) \tag{2.6}$$

In the above equation, $\mathbf{K}(\mathbf{X}^\star,\mathbf{X})$ denotes a $n^\star \times n$ matrix of the covariances evaluated at all pairs of test and train inputs points. Other matrices, $\mathbf{K}(\mathbf{X},\mathbf{X})$, $\mathbf{K}(\mathbf{X},\mathbf{X}^\star)$, $\mathbf{K}(\mathbf{X}^\star,\mathbf{X}^\star)$ are represented in similar manner. Conditioning the joint distribution in (2.6) on the observations, we obtain the predictive distribution for the Gaussian process regression,

$$p(\mathbf{f}^\star|\mathbf{y},\mathbf{X},\mathbf{X}^\star) = \mathcal{N}(\bar{\mathbf{m}},\bar{\mathbf{K}}) \tag{2.7}$$

with posterior mean function,

$$\bar{\mathbf{m}} = m(\mathbf{X}^\star) + \mathbf{K}(\mathbf{X}^\star,\mathbf{X})[\mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_\epsilon^2 I]^{-1}(\mathbf{y} - m(\mathbf{X}) \tag{2.8}$$

Assuming the prior mean function as zero, we can rewrite the predictive posterior mean as

$$\bar{\mathbf{m}} = \mathbf{K}(\mathbf{X}^\star,\mathbf{X})[\mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_\epsilon^2 I]^{-1}\mathbf{y} \tag{2.9}$$

The expression for covariance of the posterior GP is,

$$\bar{\mathbf{K}} = \mathbf{K}(\mathbf{X}^\star,\mathbf{X}^\star) - \mathbf{K}(\mathbf{X}^\star,\mathbf{X})[\mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_\epsilon^2 I]^{-1}\mathbf{K}(\mathbf{X},\mathbf{X}^\star) \tag{2.10}$$

An example Gaussian process prior is shown in Figure 2.1. We have used zero-mean and squared exponential covariance function to specify the Gaussian process. The signal variance of the SE kernel is taken as 1.2 and the value lengthscale hyperparameter is 0.3. The noise variance used in this example is 0.01. The blue line in the middle is the specified mean, and the blue-shaded region shows two standard deviations on either side of the mean. The faded "wiggly" lines are functions drawn from the prior distribution. In figure 2.2, the posterior distribution of the Gaussian

process with 12 data points is shown. We have used the same mean and covariance function specifications as before. The posterior mean function changes according to the data points as seen in the figure. The uncertainty region collapses around the data points and expands in the region as we move away from them. Similarly, functions drawn from the posterior distribution converges near to the data points, showing the confidence of the model at the data points.

Until now, we have discussed the Gaussian process regression for a scalar unknown function. But in practice it may happen that the underlying function is vector-valued, or more formally we want to make prediction for multivariate targets. In this case, we assign independent Gaussian processes to each target dimension. For example, if the latent vector-valued function has dimension $E$, we perform $E$ independent GP regressions given the train inputs for each dimension based on the data.

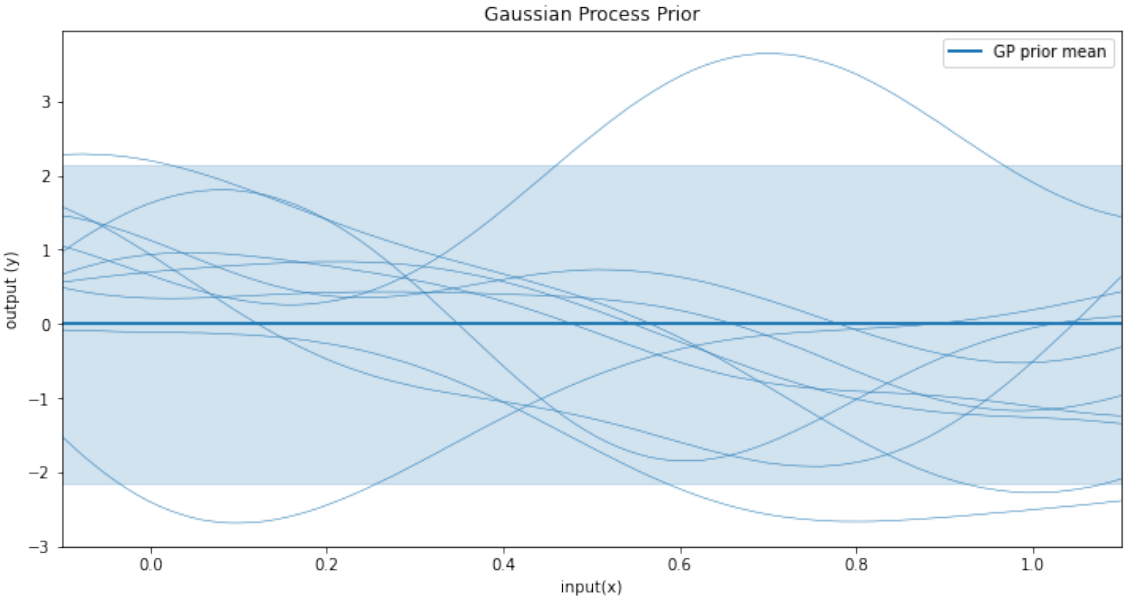


FIGURE 2.1: An example of Gaussian process prior with zero mean and squared exponential covariance function.

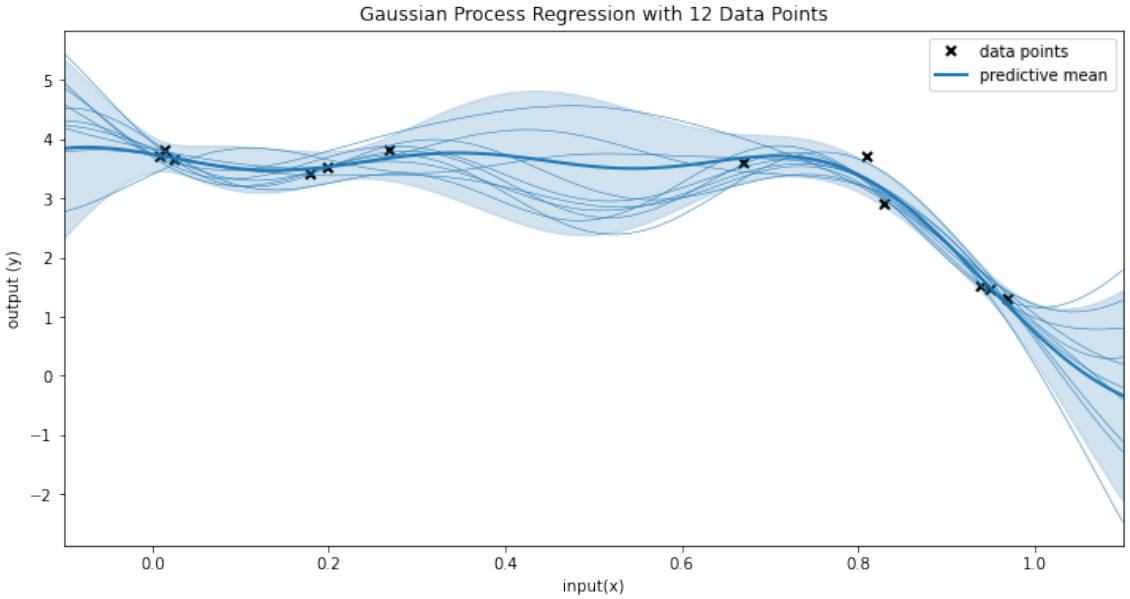


FIGURE 2.2: An example of a Gaussian process posterior distribution using 12 data points with the same mean and covariance specifications as in Figure 2.1.

## 2.4 Covariance Function

The choice of covariance function significantly determines the mathematical properties (e.g., differentiability, boundedness) of the function. Not every arbitrary function of the input pair $\mathbf{x}$, $\mathbf{x}'$ is a valid covariance function. A covariance function (also known as kernel) is said to be valid if it is positive semidefinite. A popular choice for the covariance function is the squared exponential (SE) kernel due to its suitable mathematical properties. SE kernel is infinitely differentiable, leading to a GP that has mean-square derivatives[1] of all orders. As a result the approximate function is smooth. SE kernel has the form,

$$k_{SE}(\mathbf{x}, \mathbf{x}') = \sigma_{SE}^2 \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^\intercal \Lambda^{-1}(\mathbf{x} - \mathbf{x}')\right) \tag{2.11}$$

[1]Chapter 4, Gaussian Processes in Machine Learning [58]

The above covariance function has two parameters, a scale term $\sigma^2_{SE}$, usually known as signal variance and a set of length-scales, described in terms of matrix $\Lambda$ which determines the smoothness of the function. Informally we can say length-scales determines the length of the "wiggles" in the function. The signal variance determines the variation of the function from the mean. The assumption of smoothness for SE kernel sometimes becomes unrealistic for real-world problems. Though a more suitable covariance function is the Matern [59] class of kernels for its realistic smoothness assumption, throughout this literature, we will consider the SE kernel only.

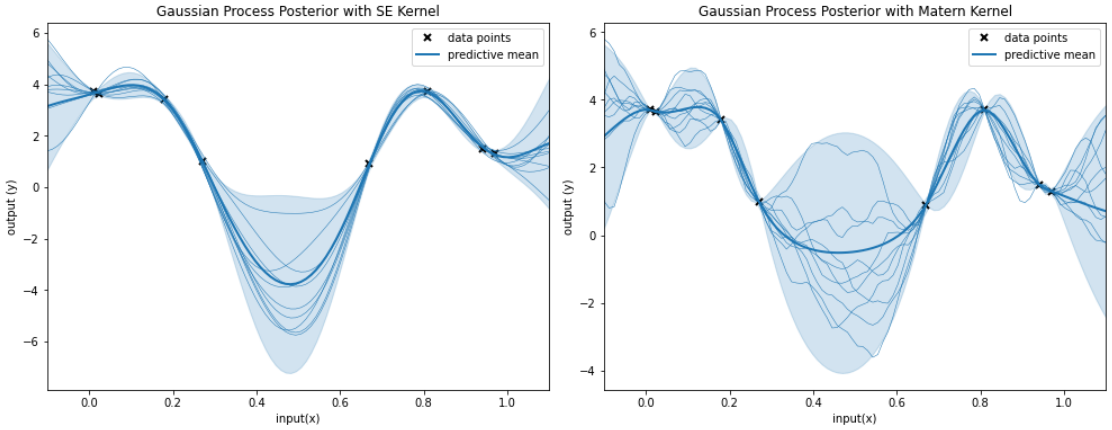


FIGURE 2.3: Gaussian process regression with SE kernel and Matern kernel.

The smoothness of a modelled function can be viewed as the quantification of similarity among neighbouring points, $\mathbf{x}$ and $\mathbf{x}'$ which provides a measure of similarity between the function values, $f(\mathbf{x})$ and $f(\mathbf{x}')$. If the neighbouring points are similar then the conditional probability of one of the points $f(\mathbf{x}')$ given $\mathbf{x}$, $\mathbf{x}'$, $f(\mathbf{x})$ will have less uncertainty. On the other hand if the neighbouring points are not similar then the same conditional probability will try to revert back to the prior probability, resulting high uncertainty around the points. In Figure 2.3 Gaussian process regression with the SE kernel and the Matern kernel on same training data set with zero prior mean function is shown. Compared to the functions drawn from the Gaussian

Process posterior with Matern kernel ("wiggly" lines on the right plot), the functions drawn from the Gaussian Process posterior ("wiggly" lines on the left plot) with SE kernel appeared to be smooth. This can be easily seen from the ripples in the sampled function from the posterior of the GP with Matern kernel.

## 2.5 Hyper-parameter Learning

In the above section, we have seen that the prediction of the Gaussian process regression depends on the choice of a covariance function from different families of it. Also, a particular class of covariance functions usually has a number of free parameters, and the shape of the latent function is highly influenced by them. These parameters are commonly known as "hyperparameters" as they parameterise the distribution of the function rather than the function itself. In Bayesian learning, choosing a covariance function for a particular application consists of selecting a setting of hyperparameters within a class and comparing across different classes. This process is usually known as "model selection" [58] in Bayesian learning. In most cases, a particular family of covariance functions is chosen beforehand, depending on the availability of prior information and desired mathematical property of the function to approximate. Then the model selection lies in finding a suitable set of hyperparameters that satisfies some predefined criterion. This process is known as the training of a GP.

We can take care of hyperparameters by finding the posterior distribution on the function values by averaging over the hyperparameters $\boldsymbol{\theta}$,

$$p(\mathbf{f}|\mathbf{y}, \mathbf{X}) = \int p(\mathbf{f}|\mathbf{y}, \mathbf{X}, \boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} \tag{2.12}$$

where $p(\boldsymbol{\theta})$ is the prior distribution over the hyperparameters. Solving the integration in Eq. (2.12) is complex and may not be analytically tractable. One can use an approximate inference method like Markov Chain Monte Carlo (MCMC) or variational inference or can maximize the marginal likelihood of the data as given below with respect to the hyperparameters $\boldsymbol{\theta}$,

$$p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{X}) = \int p(\mathbf{y}|\mathbf{f}, \mathbf{X}, \boldsymbol{\theta}) p(\mathbf{f}|\mathbf{X}, \boldsymbol{\theta}) d\mathbf{f} \tag{2.13}$$

This approximation approach is known as type-II maximum likelihood (ML) method for model selection. Eq. (2.13) is analytically solvable with Gaussian noise assumption that gives a Gaussian likelihood of the data. The log-marginal likelihood can then be written as,

$$\begin{aligned}\log p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{X}) = -\frac{1}{2}(\mathbf{y} - m(\mathbf{X}))^{\intercal}[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_{\epsilon}^2 I]^{-1}(\mathbf{y} - m(\mathbf{X})) \\ + \frac{1}{2} \mid \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_{\epsilon}^2 I \mid - \frac{N}{2} \log 2\pi\end{aligned} \tag{2.14}$$

In Eq. (2.14) the first term is known as data fit term as it contains the observations, the second term is known as the model complexity term, and the last term is a normalizing constant. The model complexity term penalizes the optimization objective depending upon the covariance function. Partial derivatives of the log-marginal likelihood w.r.t. the hyperparameters can be found easily. We can use gradient based optimization method to find the optimal value of them, though global optima is not guaranteed. One can argue that ML-II optimization may lead to over-fitting of the data, causing generalization error, but it is not a big problem as we are optimizing w.r.t. the hyperparameters to adapt the distribution over the function rather than finding the best fit of the function.

Examples of Gaussian process regression with varying length-scale hyperparameter

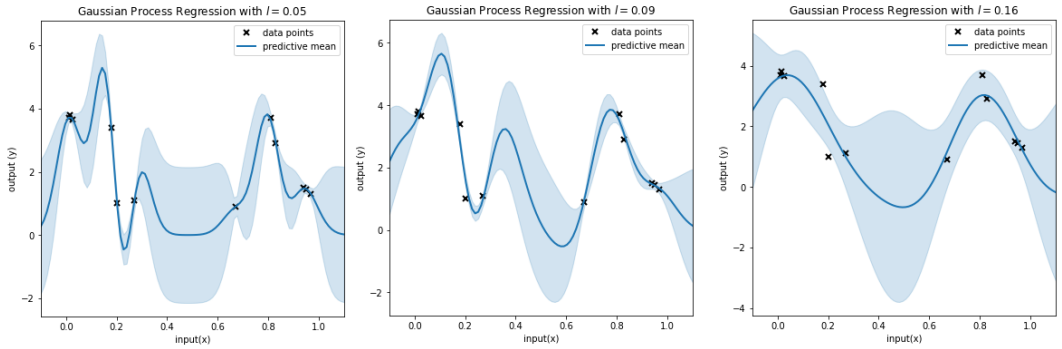


FIGURE 2.4: Gaussian process regression with SE kernel and different lengthscales

of SE kernel is shown in Figure 2.4. We have considered three cases. In all of these cases the values of other two hyperparameters (signal variance =1.2 and noise variance $^2$=0.3) are kept fixed. The values of the lengthscale hyperparameter for the three plots from left to right are $l = 0.05$, $l = 0.09$, $l = 0.16$, respectively. As we move from left to right the correlation between functional values decreases. The right most plot corresponds to the optimized lengthscale hyperparameter.

## 2.6 Prediction at Uncertain Inputs

In section 2.3 we have seen the expression for the predictive mean and variance of the GP posterior distribution at known input points $\mathbf{x}^\star$. However it may happen that the test inputs are uncertain, we only know the distribution of it. For this case, the predictive distribution can be found by averaging over the uncertainty in the input point,

$$p(f^\star|\mathbf{y}, \mathbf{X}) = \int p(f^\star|\mathbf{y}, \mathbf{X}, \mathbf{x}^\star)p(\mathbf{x}^\star)d\mathbf{x}^\star \tag{2.15}$$

[2]Sometimes the noise variance, $\sigma_\epsilon^2$ is not considered is a hyperparameter, though it plays an analogous role and can be treated as a hyperparameter. For more details check [58]

This new predictive distribution, $p(f^\star|\mathbf{y}, \mathbf{X})$ is known as the marginal predictive distribution, as it is obtained by marginalizing the predictive distribution with respect to the test input $\mathbf{x}^\star$.

A common assumption about the uncertain test inputs is that they are distributed as Gaussian with known mean and variance, $\mathbf{x}^\star \sim \mathcal{N}(\boldsymbol{\mu}^\star, \boldsymbol{\Sigma}^\star)$. As a Gaussian input, $\mathbf{x}^\star$ is mapped through a nonlinear function, the resulting predictive distribution will not be Gaussian as shown in Figure 2.5. Computing the analytical integration of Eq. (2.15) becomes impossible as it can not be easily parameterized in terms of any

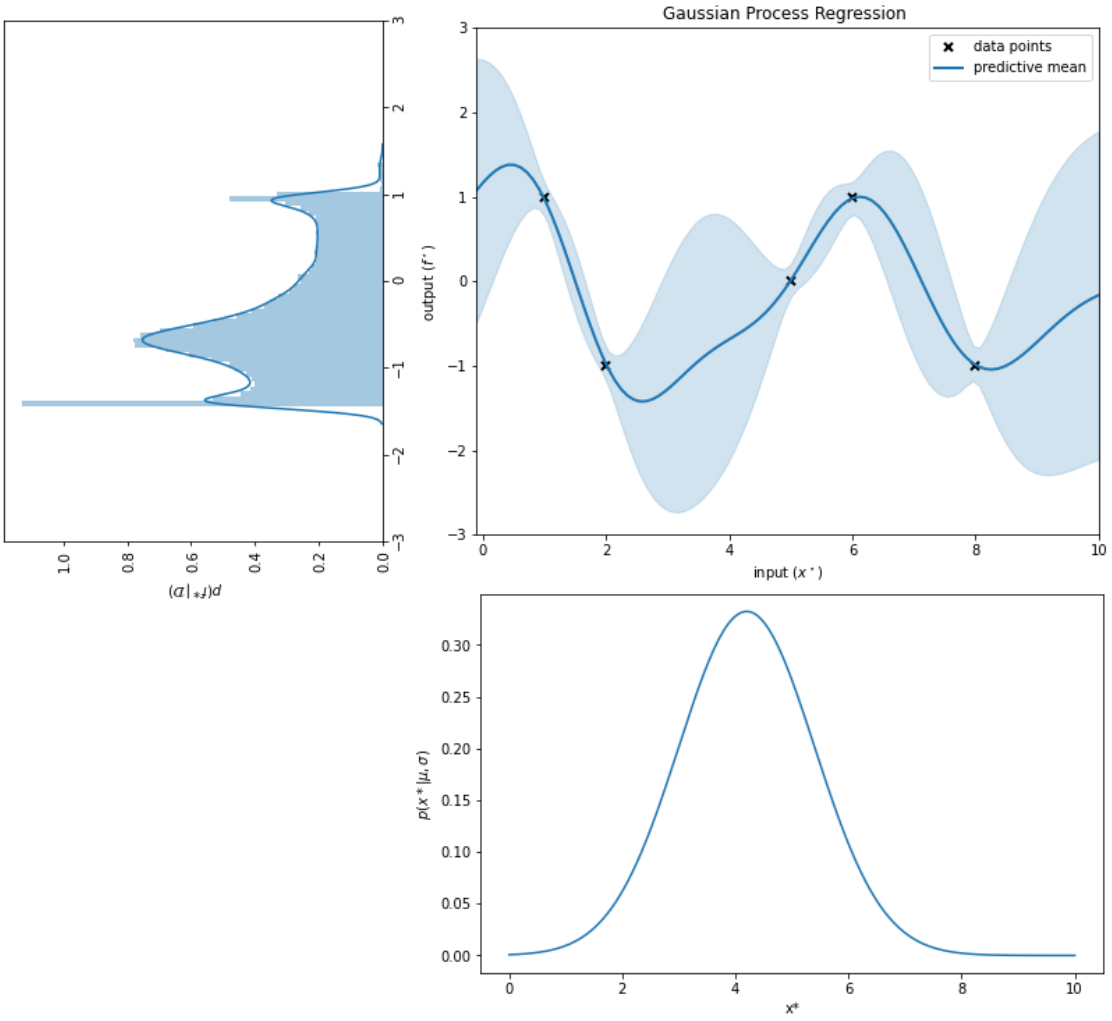


FIGURE 2.5: An example of GP prediction at a Gaussian test input.

standard distributions. There are different approaches to approximate the posterior distribution. We will discuss some of them in the following sections.

### 2.6.1 Numerical Approximation

The predictive distribution can be found by performing a numerical approximation of the integral in Eq. (2.15) using simple Monte Carlo approach [60],

$$p(f^\star|\mathbf{y},\mathbf{X}) \simeq \frac{1}{T}\sum_{t=1}^{T} p(f^\star|\mathbf{y},\mathbf{X},\mathbf{x}^{\star t}) \tag{2.16}$$

where, $\mathbf{x}^{\star t}$ are independent samples drawn from $p(\mathbf{x}^\star)$. In Figure 2.5 prediction of a Gaussian process regression at a test input distributed according to the bottom plot is shown. The top right plot shows a GP posterior based on the training data. The blue histogram at top left shows the true marginal predictive distribution. To generate the true marginal predictive distribution, 100,000 samples are drawn from the input distribution. These input samples are then propagated through the model trained with the available data. These 100,000 predicted values are used to estimate the marginal predictive density.

Even though the integration of Eq.(2.15) may be calculated using sampling-based techniques, it might be beneficial to identify a parametric expression for the posterior distribution. Also, sampling based techniques are usually time consuming and not suitable for fast systems. To find an approximate analytic solution of the integration, the marginal predictive distribution is approximated with a Gaussian distribution. For this case, the problem becomes the computation of the mean and variance of the approximated Gaussian distribution.

### 2.6.2 Exact Moment Matching

In exact moment matching [42, 61] method the predictive distribution is approximated by a Gaussian that possesses the same mean and variance of the true predictive distribution, thus known as exact moment matching. The predictive mean can be found by the law of iterated expectation,

$$\begin{aligned}\mathbb{E}[f^\star|\mathbf{y},\mathbf{X}] &= \mathbb{E}_{x^\star\sim p(x^\star)}\left[\mathbb{E}_{f^\star\sim p(f^\star|\mathbf{y},\mathbf{X},\mathbf{x}^\star)}[f^\star|\mathbf{x}^\star]|\boldsymbol{\mu}^\star,\boldsymbol{\Sigma}^\star\right]\\ &= \mathbb{E}_{x^\star}[m(\mathbf{x}^\star)|\boldsymbol{\mu}^\star,\boldsymbol{\Sigma}^\star] + \mathbb{E}_{x^\star}[\bar{m}(\mathbf{x}^\star)|\boldsymbol{\mu}^\star,\boldsymbol{\Sigma}^\star]\\ &= \mathbb{E}_{x^\star}[m(\mathbf{x}^\star)|\boldsymbol{\mu}^\star,\boldsymbol{\Sigma}^\star] + \mathbb{E}_{x^\star}[\mathbf{k}(\mathbf{x}^\star,\mathbf{X})|\boldsymbol{\mu}^\star,\boldsymbol{\Sigma}^\star]\boldsymbol{\beta}\end{aligned} \tag{2.17}$$

where $\boldsymbol{\beta} = [\mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_\epsilon^2 I]^{-1}(\mathbf{y} - \mathbf{m}(\mathbf{X}))$. In the above expression, the expectation over the covariance function can be computed in closed form for squared exponential kernel. Similarly, predictive variance can be found using the law of total variance,

$$\begin{aligned}&\mathbb{V}[f^\star|\mathbf{y},\mathbf{X}]\\ &= \mathbb{V}_{x^\star\sim p(x^\star)}\left[\mathbb{E}_{f^\star\sim p(f^\star|\mathbf{y},\mathbf{X},\mathbf{x}^\star)}[f^\star]|\boldsymbol{\mu}^\star,\boldsymbol{\Sigma}^\star\right] + \mathbb{E}_{x\sim p(x^\star)}\left[\mathbb{V}_{f^\star\sim p(f^\star|\mathbf{y},\mathbf{X},\mathbf{x}^\star)}[f^\star]|\boldsymbol{\mu}^\star,\boldsymbol{\Sigma}^\star\right]\\ &= \mathbb{V}_{x^\star}[m(\mathbf{x}^\star)|\boldsymbol{\mu}^\star,\boldsymbol{\Sigma}^\star] + 2\mathbb{C}\left[m(\mathbf{x}^\star),\mathbf{k}(\mathbf{x}^\star,\mathbf{X})\right]\boldsymbol{\beta} + \boldsymbol{\beta}^\intercal\mathbb{V}_{x^\star}\left[\mathbf{k}(\mathbf{x}^\star,\mathbf{X})\right]\boldsymbol{\beta} +\\ &\qquad \mathbb{E}\left[k(\mathbf{x}^\star,\mathbf{x}^\star)\right] - \mathbb{E}\left[\mathbf{k}(\mathbf{x}^\star,\mathbf{X})\left[\mathbf{K} + \sigma_\epsilon^2 I\right]^{-1}\mathbf{k}(\mathbf{X},\mathbf{x}^\star)\right]\end{aligned} \tag{2.18}$$

In Eq. (2.18) the variance and covariance terms can be computed in closed form for the squared exponential kernel. The above expressions are derived for a one-dimensional regression problem. An extension to the multivariate regression problem is given in [32].

### 2.6.3 Mean Equivalent Approximation

It is a simple and computationally cheap method to compute the marginal predictive distribution. In this method we only consider the mean of the input distribution and make prediction with respect to it.

$$\mathbb{E}[f^\star|\mathbf{y},\mathbf{X}] = \bar{m}(\boldsymbol{\mu}^\star) \tag{2.19}$$

$$\mathbb{V}[f^\star|\mathbf{y},\mathbf{X}] = \bar{\mathbf{K}}(\boldsymbol{\mu}^\star) \tag{2.20}$$

Mean equivalent approximation neglects the uncertainty in the input distribution which induces model error. This leads to poor approximations for iterative multiple step ahead prediction. In [42] it was demonstrated that with increase in prediction horizon model errors will accumulate to give larger error for later stage of prediction.

### 2.6.4 Linearization of the Posterior GP Mean Function

An alternative way to find the predictive distribution is to linearize the posterior GP mean function [43] in Eq.(2.9) around the mean of the test input distribution. Using results for mapping a Gaussian distribution through linear models, we get the moments of the approximated posterior Gaussian distribution. The predictive mean is computed by evaluating the posterior GP mean in Eq.(2.9) at $\boldsymbol{\mu}^\star$, i.e.,

$$\mathbb{E}[f^\star|\mathbf{y},\mathbf{X}] = \bar{m}(\boldsymbol{\mu}^\star) \tag{2.21}$$

The expression for predictive variance is given by,

$$\mathbb{V}[f^\star|\mathbf{y},\mathbf{X}] = \bar{\mathbf{K}}(\boldsymbol{\mu}^\star) + \nabla_{\mathbf{x}^\star}\bar{m}(\boldsymbol{\mu}^\star)\boldsymbol{\Sigma}^\star\nabla_{\mathbf{x}^\star}\bar{m}(\boldsymbol{\mu}^\star)^\intercal \tag{2.22}$$

The derivation of the above expression is shown in A.1. Though the moment matching method gives better approximation of the mean and variance of the marginal predictive distribution, its application is limited due to the requirement of some expensive computations. The computational complexity of moment matching for a single time step is $\mathcal{O}(n^2E^2D)$, where $n$ is the number of GP training points, $D$ is the input dimension and $E$ is the dimension of the latent function. On the other hand the computational complexity of linearizing the posterior GP mean function is $\mathcal{O}(n^2ED)$ for a single time step. As we can see, linearizing the posterior mean function is relatively computationally cheaper as the number of output dimensions increases, resulting in faster computation. Sometimes, moment matching may also induce model error as the true predictive distribution may not be Gaussian or unimodal. Both approaches scale directly with input and output dimensions, as well as with the number of training data points, making them computationally expensive in high-dimensional spaces.

In Figure 2.6 the predictive distribution of Gaussian process regression with Gaussian test input (bottom plot) using different approximation methods is shown (top left). With all three methods, we are approximating the true predictive distribution with a Gaussian, although the true predictive distribution may not be unimodal. As we can see, the first and second moments of the predictive distribution computed using the moment matching method matched closely with the true distribution. These two still differ in shape that induces prediction error. On the other hand, linearizing the posterior GP mean method works well locally and tends to capture the characteristics near the mean of input test point but underestimate the variance when the true predictive distribution is not unimodal as shown in the figure.

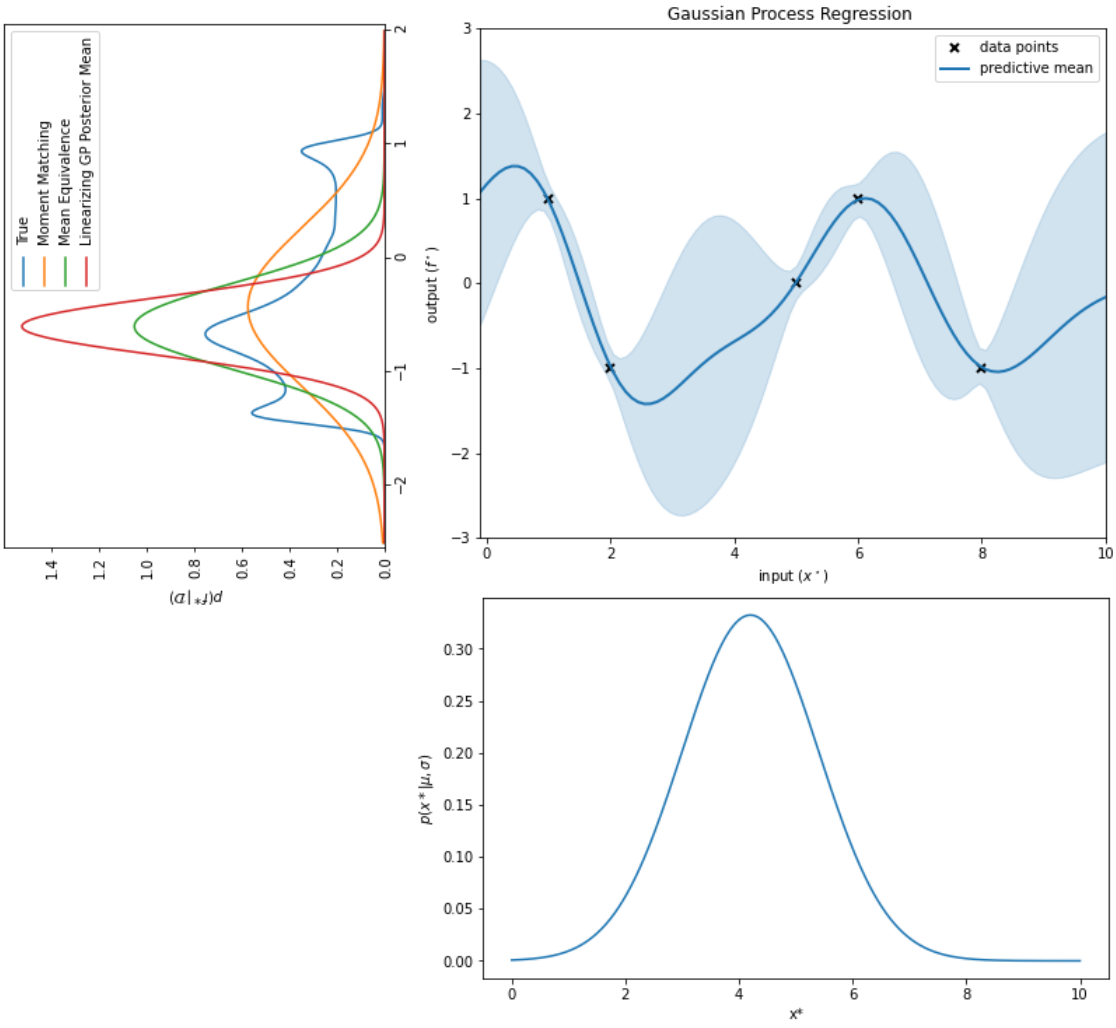


FIGURE 2.6: GP marginal predictive distribution at a Gaussian test input using different approximation methods.

## 2.7 Gaussian Process for Modeling Dynamical System

In section 1.1.1 we have seen that dynamical systems play a key role in control. Depending on the complexity, analytical model of the system may or may not be available. Gaussian process regression can be used to model the unknown dynamics of the system from available data. We consider a discrete-time, continuous-state dynamical system with latent state $\mathbf{x}_t$ at time $t$. We consider that we have access

to a sequence of noisy measurements $\{\mathbf{y}_1, \mathbf{y}_2, .....\mathbf{y}_T\}$. In addition, we assume that control input $u_t$ can be applied to drive the system to a desired state. This control input may be a function of noisy measurement $y_t$. We wish to learn a model of the system using these information.

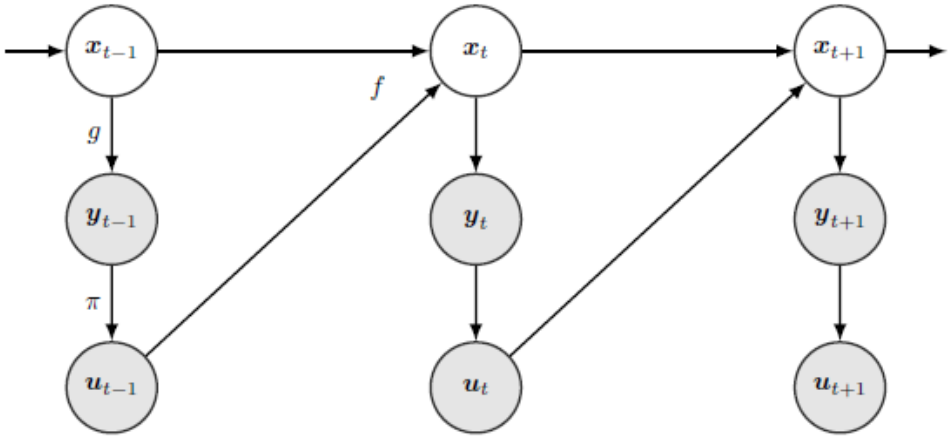


FIGURE 2.7: Graphical representation of learning dynamical system

Fig. 2.7 shows a graphical representation of the above-mentioned problem [61]. At every time step we make observations $y$ of the latent variable $x$ through a function $g$ that may be unknown. The control input $u$ depends on the observation through a policy $\pi$. The transition dynamics between the hidden states is represented by a nonlinear unknown function $f$ which depends on both the previous state and the control.

### 2.7.1 Gaussian Process State-space Model

As discussed in section 2.3, GP regression models can be used to learn an unknown function from data with little prior knowledge of the system. It captures the uncertainty within the model class by returning a probability distribution over the

unknown function rather than a point estimate. In our case we place a GP prior on the unknown transition function $f(x_t)$.

$$f \sim \mathcal{GP}\left(m, k\right) \tag{2.23}$$

As mentioned before, zero-mean GP prior with squared exponential kernel is our choice for the regression problem. We introduce a random variable $f_{t+1}$ to represent the GP function evaluated at $x_t$,

$$f_{t+1} = f\left(x_t\right) \tag{2.24}$$

$$y_t = x_t + \epsilon_t \tag{2.25}$$

For a system with control input, $f(x_t, u_t)$, we use augmented state control vector $\hat{\mathbf{x}}_t$ to represent the feature vector in our regression problem.

$$\hat{\mathbf{x}}_t = [x_t^\intercal \;\; u_t^\intercal]^\intercal \tag{2.26}$$

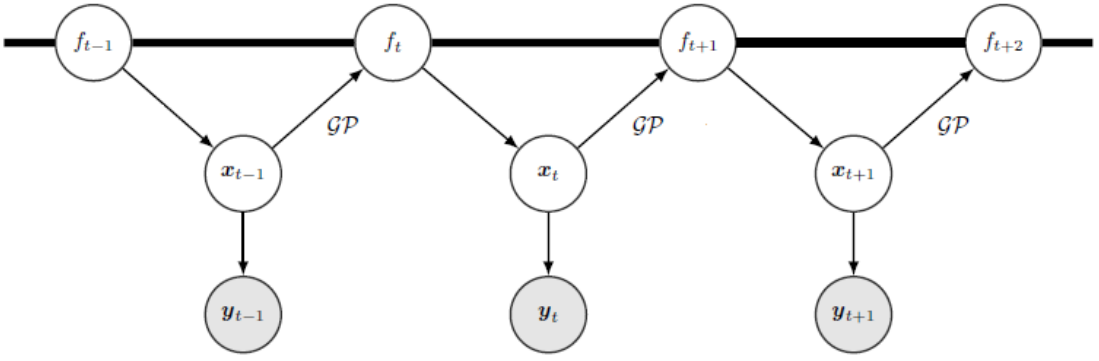


FIGURE 2.8: Graphical representation of Gaussian process state space model

Figure 2.8 shows the graphical representation of Gaussian process state space model

where the random variables $f$ represent GP functions. Notice that the GP functions are connected fully to each other as represented by the black thick line. For the $D$ dimensional state space, we use $D$ independent GPs to model the mapping from current states and control inputs to each state at the next time instant.

$$f^1 \sim \mathcal{GP}\left(\mathbf{m}_1(\hat{\mathbf{x}}_t), k_1(\hat{\mathbf{x}}_t^i, \hat{\mathbf{x}}_t^j)\right) \tag{2.27}$$

$$f^2 \sim \mathcal{GP}\left(\mathbf{m}_2(\hat{\mathbf{x}}_t), k_2(\hat{\mathbf{x}}_t^i, \hat{\mathbf{x}}_t^j)\right)$$
$$\vdots$$
$$f^D \sim \mathcal{GP}\left(\mathbf{m}_D(\hat{\mathbf{x}}_t), k_D(\hat{\mathbf{x}}_t^i, \hat{\mathbf{x}}_t^j)\right)$$

Notice that in the above expressions (2.27) each independent GP model has a separate set of hyperparameters for the covariance function, although their training features are the same. Next we will see an example of Gaussian process regression for learning transition dynamics of a system. We consider the Van der Pole equation given in [62] with the parameter value, $\varepsilon = 1$. We use Gaussian process state space model to learn the unknown dynamics from the data. $n =$ data points are generated using the true dynamics with additive Gaussian noise. The learned model is used to predict the function in future. The state trajectories and phase plots are shown in Figure 2.9, 2.10 and 2.11 respectively. We used two independent GPs to model the transition dynamics of the two states.

For the first case (left plots) we used $n = 50$ noisy data points along the trajectory for training the GPs, while for the second case (right plots) we used $n = 100$ noisy data points along the trajectory. The blue-shaded region in the state trajectory plots shows the uncertainty in the model. For the first case, the GP model produces good result near the data points. It induces errors while predicting in the region beyond

the current data points, delivering higher uncertainty. This uncertainty measure is very useful for learning system dynamics as it reflects our confidence on the model. As we collect more data model uncertainty reduces improving the performance of the model which can be seen from the right plots. The phase plot for two different cases is shown in Figure 2.11. As we can see there is a significant deviation from the true trajectory at the region with no data points. For the right plot the deviation reduces converging to the true trajectory, which shows that GP regression models are capable of capturing complex nonlinear patterns within the data.

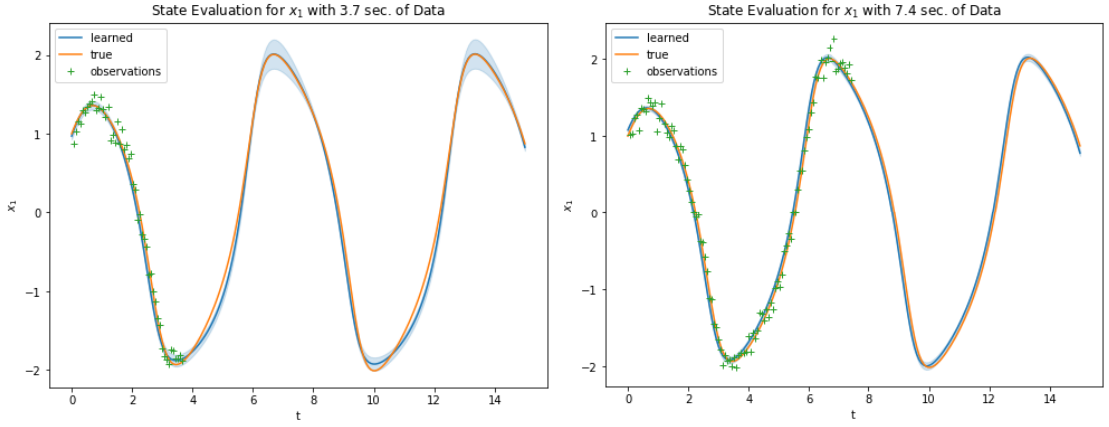


FIGURE 2.9: State trajectory of Van der Pol oscillator: $x_1(t)$; left plot: 50 training data points; right plot: 100 training data points.

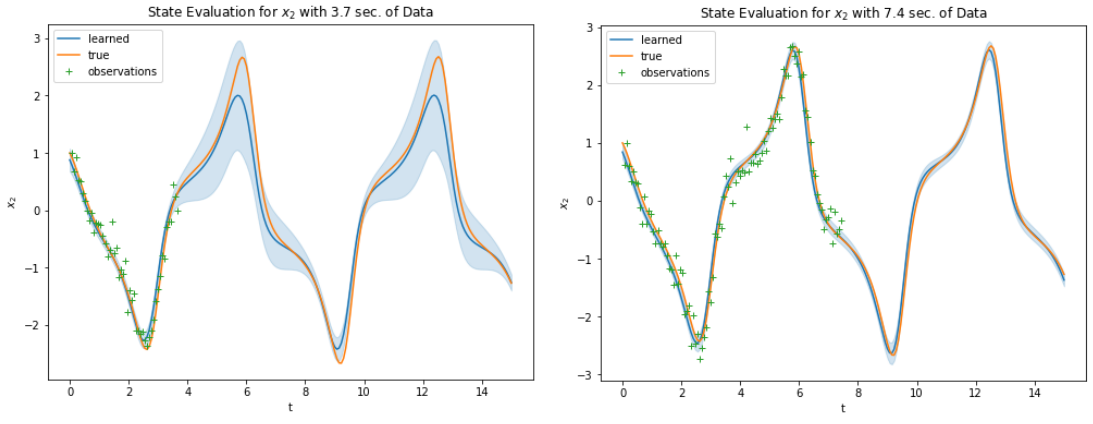


FIGURE 2.10: State trajectory of Van der Pol oscillator:$x_2(t)$; left plot: 50 training data points; right plot: 100 training data points.

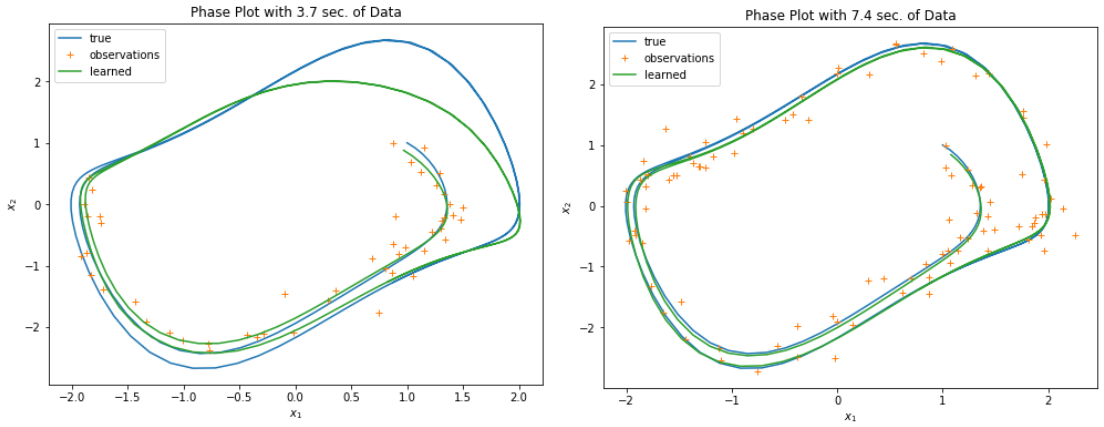


FIGURE 2.11: Phase portraits of Van der Pol oscillator; left plot: 50 training data points; right plot: 100 training data points.

### 2.7.2 Online Learning and Limitations

As a non-parametric model, Gaussian processes use the whole data set while making a prediction. With a larger data set, the computational complexities of GP training and prediction becomes high, limiting its application to smaller data sets. For a data set of size $n$, training a GP using gradient-based evidence maximization (Eq.2.14) requires $\mathcal{O}(n^3)$ computations. The reason behind it is that at each step we have to find the inversion of covariance matrix $\mathbf{K}_{nn}$. With $E$ target dimensions, the complexity becomes $\mathcal{O}(En^3)$ for GP training.

For prediction at uncertain input with linearization of posterior mean function, with $D$ input dimensions and $E$ target dimensions the computational complexity becomes $\mathcal{O}(n^2ED)$ for a single time step (for exact moment matching it is $\mathcal{O}(n^2E^2D)$). For online learning we update the training data set with new measurements. Thus, the size of training data set grows with time. Every time, a new measurement comes we have to train the GP model on the entire data set (the new data point is augmented to the old data set) making the use of GP models prohibitive in online learning scenario.

In the scalable GP literature, there are two main approaches for addressing this issue. One approximates the kernel matrix globally by removing unnecessary information from the data set and kernel matrix. This kind of global approximations can be achieved by (i) forming a subset of training data with $m \ll n$ points [63]. This gives a smaller, but constant, size kernel matrix $\mathbf{K}_{mm}$; (ii) forming a sparse kernel matrix [64] by removing the uncorrelated entries of $\mathbf{K}_{nn}$; (iii) Using inducing point methods resulting in Nystrom approximation of the kernel matrix $\mathbf{K}_{nn} \approx \mathbf{K}_{nm}\mathbf{K}_{mm}\mathbf{K}_{mn}$ [55, 65]. Similar to the regression data matrix $\mathbf{X}$ defined in 2.3, the inducing point matrix is defined and denoted as $\mathbf{U}$. The other approach uses the divide-and-conquer rule to find a local approximate of the training data [66, 67]. Global approximations are capable of capturing global patterns, though they often filter out local information because of the limited global inducing set. Local approximations give better results by focusing on the local patterns of the data set, but sometime suffer from local overfitting and risk of discontinuous prediction. A detailed discussion of these methods can be found in [68].

Some of the sparse approximation methods mentioned above can be applied to the online learning scenario. The model should be able to adapt to the real time data arriving sequentially. Also, it has to be scalable to large data sets as new data comes continuously. [69, 70] extends the sparse GP using induction points approach to online learning but sacrifices some performance by fixing hyperparameters to get constant time update. A framework for deploying Gaussian process probabilistic models in streaming data setting was developed in [71]. This presented two probabilistic approaches (O-SVGP and O-SGPR) to update posterior distribution and the hyperparameters in online manner. However, as shown in [72] these approaches suffer from generalization error. [72] presented an approach using structured kernel interpolation [73] to achieve constant time online update with respect to the size

of the data set $n$. We will discuss online structured kernel interpolation [72] in the next section.

## 2.8 Woodbury Inversion for Structured Kernel Interpolation (WISKI)

Structured kernel interpolation (SKI) generalizes the inducing point methods for scalable Gaussian processes. It provides a unified way for fast kernel approximation through kernel interpolation. WISKI [72] combines SKI and the Woodbury identity to provide online learning with a constant time update in $N$. SKI approximates the $N \times M$ cross-covariance matrix ($\mathbf{K}_{XU}$) evaluated at each of $N$ training and $M$ inducing point pairs [55] by interpolating the $M \times M$ covariance matrix $\mathbf{K}_{UU}$.

$$\mathbf{K}_{XU} \approx \mathbf{W}\mathbf{K}_{UU} \tag{2.28}$$

where W is a $N \times M$ sparse matrix of interpolation weights. Similarly, $\mathbf{K}_{XX}$ can be approximated as

$$\mathbf{K}_{XX} + \sigma_\epsilon^2\mathbf{I} \approx \tilde{K}_{XX} + \sigma_\epsilon^2\mathbf{I} \approx \mathbf{K}_{XU}\mathbf{K}_{UU}\mathbf{K}_{UX} + \sigma_\epsilon^2\mathbf{I} \approx \mathbf{W}\mathbf{K}_{UU}\mathbf{W}^\intercal + \sigma_\epsilon^2\mathbf{I} \tag{2.29}$$

The above expression is the building block of the online Gaussian process approximation. Though Eq.(2.29) offers a fast approximation of the kernel matrix, we are more interested in finding the inverse. Using the Woodbury matrix identity (a special case of matrix inversion lemma) to the inverse of the above expression, the SKI

kernel inverse can be written as

$$\left(\tilde{\mathbf{K}}_{XX} + \sigma_\epsilon^2 \mathbf{I}\right)^{-1} = \frac{1}{\sigma_\epsilon^2}\mathbf{I} - \frac{1}{\sigma_\epsilon^2}\mathbf{W}\mathbf{M}\mathbf{W}^\intercal \tag{2.30}$$

where $\mathbf{M} = (\sigma_\epsilon^{-2}\mathbf{K}_{UU} + \mathbf{W}^\intercal\mathbf{W})^{-1}$. If we consider that the above information is available after $t$ data points, then after observing $(t+1)^{th}$ data point $\mathbf{M}$ can be updated by a rank one update,

$$\mathbf{M}_{t+1}^{-1} = \mathbf{M}_t^{-1} + \mathbf{w}_{t+1}\mathbf{w}_{t+1}^\intercal \tag{2.31}$$

where, $\mathbf{w}_{t+1}$ is the interpolation vector for the $(t+1)^{th}$ data point.

### 2.8.1 Computing the Moments of Posterior Distribution

Substituting Eq.(2.30) in Eq.(2.9), Eq.(2.10) and Eq.(2.14) we get the following expression for posterior mean (Eq.(2.32)), posterior covariance function (Eq.(2.33)) and marginal log-likelihood (Eq.(2.34)).

$$\bar{\mathbf{m}} = \mathbf{w}_{\mathbf{x}_\star}^\intercal \mathbf{M}\mathbf{W}^\intercal\mathbf{y} \tag{2.32}$$

$$\bar{k}(\mathbf{x}_{\star_i}, \mathbf{x}_{\star_j}) = \sigma_\epsilon^2 \mathbf{w}_{\mathbf{x}_{\star_i}}^\intercal \mathbf{M}\mathbf{w}_{\mathbf{x}_{\star_j}} \tag{2.33}$$

$$\begin{aligned}\log(p(\mathbf{y}|\mathbf{x},\boldsymbol{\theta})) = &-\frac{1}{2\sigma_\epsilon^2}\left(\mathbf{y}^\intercal\mathbf{y} - \mathbf{y}^\intercal\mathbf{W}\mathbf{M}\mathbf{W}^\intercal\mathbf{y}\right) \\ &- \frac{1}{2}\left(\log(\mathbf{K}_{UU}) - \log(|\mathbf{M}|) + (N-M)\log\sigma_\epsilon^2\right)\end{aligned} \tag{2.34}$$

In the above expression, $\mathbf{w}_{\mathbf{x}_\star}$ represents the interpolation vector corresponding to the test input. Computing Eq.(2.31) requires computation of $\mathbf{K}_{UU}^{-1}$. But sometimes

the inverse of $\mathbf{K}_{UU}$ will be ill-conditioned. For this case we need an approach that uses $\mathbf{K}_{UU}$ directly in the predictions. The above interpolation matrix can be approximated as $\mathbf{W}^\intercal\mathbf{W} = \mathbf{L}\mathbf{L}^\intercal$ as shown in [72]. Substituting this in Eq.(2.30) we get the following equations.

$$\mathbf{M} = \sigma_\epsilon^{-2}\mathbf{K}_{UU} - \sigma_\epsilon^{-2}\mathbf{K}_{UU}\mathbf{L}\mathbf{Q}^{-1}\mathbf{L}^\intercal\sigma_\epsilon^{-2}\mathbf{K}_{UU} \tag{2.35}$$

$$\mathbf{Q} = \mathbf{I} + \mathbf{L}^\intercal\sigma_\epsilon^{-2}\mathbf{K}_{UU}\mathbf{L} \tag{2.36}$$

Using the above expressions the predictive mean, variance and the marginal likelihood can be formulated as,

$$\bar{\mathbf{m}} = \mathbf{w}_{\mathbf{x}_\star}^\intercal \left(\sigma_\epsilon^{-2}\mathbf{K}_{UU}(\mathbf{W}^\intercal\mathbf{y} - \mathbf{L}\mathbf{b})\right)\mathbf{W}^\intercal\mathbf{y} \tag{2.37}$$

$$\bar{k}(\mathbf{x}_{\star_i}, \mathbf{x}_{\star_j}) = \sigma_\epsilon^2\mathbf{w}_{\mathbf{x}_{\star_i}}^\intercal\left(\mathbf{K}_{UU}(\mathbf{w}_{\mathbf{x}_{\star_j}} - \mathbf{L}\mathbf{b})\right) \tag{2.38}$$

$$\begin{aligned}\log(p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})) = -\frac{1}{2\sigma_\epsilon^2}\left(\mathbf{y}^\intercal\mathbf{y} - \mathbf{y}^\intercal\mathbf{W}\mathbf{K}_{UU}\mathbf{W}^\intercal\mathbf{y} + \mathbf{a}^\intercal\mathbf{Q}^{-1}\mathbf{a}\right) \\ - \frac{1}{2}\left(-\log|\mathbf{Q}| + (N-M)\log\sigma_\epsilon^2\right)\end{aligned} \tag{2.39}$$

In the above expression we can cache $\mathbf{W}^\intercal\mathbf{y}$, $\mathbf{W}^\intercal\mathbf{y}$ and $L$. When new data arrives a single updation step step can be perform to update these values. In the following expressions $(\mathbf{x}_{t+1}, y_{t+1})$ is the new data point available at time $t+1$. Similarly, $\mathbf{w}_{t+1}$ represents the weight vector corresponding to new data point.

$$(\mathbf{W}^\intercal\mathbf{y})_{t+1} = (\mathbf{W}^\intercal\mathbf{y})_t + y_{t+1}\mathbf{w}_{x_{t+1}} \tag{2.40}$$

$$(\mathbf{y}^\intercal\mathbf{y})_{t+1} = (\mathbf{y}^\intercal\mathbf{y})_t + y_{t+1}^2 \tag{2.41}$$

$$(\mathbf{W}^\intercal \mathbf{W})_{t+1} = (\mathbf{W}^\intercal \mathbf{W})_t + \mathbf{w}_{x_{t+1}} \mathbf{w}_{x_{t+1}}^\intercal \tag{2.42}$$

## 2.9 Summary

Machine learning techniques are very useful for learning a model of the system from data when few or no prior information of the system is available. The learned model converges to the true system model in presence of infinite amount of data. In smaller data region, these methods suffer from model generalization error. Bayesian machine learning techniques work well in smaller data region as it quantifies the uncertainty within the model. Gaussian processes are very popular in Bayesian learning paradigm for learning a model with limited amount of data. It gives a probabilistic interpretation of the data and quantifies the uncertainty within the prediction. A Gaussian process can be fully specified by its mean and covariance functions. Usually, the mean function of a Gaussian Process is considered zero. Covariance function provides a measure of similarity between two points. Squared exponential kernel is a very popular choice for covariance function as functions modelled with this are very smooth. Finding a closed form expression for predictive mean and covariance function of Gaussian Process posterior at uncertain test point (which can be represented with probability distribution ) is not possible. Moreover, the predictive distribution may not be unimodal. A very effective approach to find the predictive mean and variance is to approximate the posterior distribution by a Gaussian distribution. Linearization of the posterior mean function around the mean of the uncertain test point gives a good approximation of the true posterior distribution. This approach is computationally cheap also.

Though Gaussian Processes are very powerful in presence of limited amount of data, it suffers when the number of training data points increases as it involves the computation of the inverse of the covariance matrix. For learning a GP model from sequential data, it has to be trained every time when a new data point is available. This is computationally expensive as the size of the data set increases with time. Structured kernel interpolation technique with Woodbury inversion gives us the freedom to update the posterior mean and variance of the predictive distribution, when a new data point is available.

# Chapter 3

# Online Learning Based MPC

## 3.1 Introduction

In online learning-based model predictive control, we learn the unknown part of system dynamics from the streaming data (noisy measurements) and use the learned model to plan the future behavior. We use the structured kernel interpolation method for online Gaussian process regression as discussed in Section 2.8 for efficient and online update of the model with new measurements. We use some of the theory discussed in Section 2 in this section.

## 3.2 MPC Controller Design

We can divide this approach into three parts. The first part is learning a Gaussian process prediction model from the streaming data. The model has to be updated every time when a new measurement is available. The next step is the selection of a suitable cost function and the last step lies in the formulation of chance constraints

for tractable MPC optimization. In this section, we use $k$ to denote an instant of time and $i = 0, 1, 2, ...$ represents future state starting at a particular time instant $k$. We consider the design of model predictive controller for dynamical system discussed in section 1.1.1. We assume that the nominal model of the system is known and the system dynamics can be represented by Eq.(1.2). Further we consider that the unknown part of the dynamical system $\boldsymbol{g}$ lies in a subspace spanned by $\mathbf{B}_d$ as described below.

$$\mathbf{x}_{k+1} = \boldsymbol{h}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{B}_d \left(\boldsymbol{g}(\mathbf{x}_k, \mathbf{u}_k) + \boldsymbol{\epsilon}_k\right) \tag{3.1}$$

with states $\mathbf{x} \in \mathbb{R}^{n_x}$ and control $\mathbf{u} \in \mathbb{R}^{n_u}$. This is a common assumption for control system design, and later we will see that this assumption significantly reduces the computational burden by giving us the flexibility to select a subset of state space that is assumed to be affected by uncertainty, as shown in [51]. The dimension of $\mathbf{B}_d$ depends on our choice of uncertain states. For example, if we want to learn $n_d$ states then dimension of $\mathbf{B}_d$ will be $n_x \times n_d$. In the above equation, $\boldsymbol{\epsilon}$ represents the measurement noise as we are interested in estimating the unknown function $\boldsymbol{g}$ from noisy, sequential observations of states and control. For this, we apply structured kernel interpolation for online Gaussian process regression (WISKI) discussed in section 2.8 to learn a statistical model of the unknown function $\boldsymbol{g}$ from available data. At each time instant, the learned GP model is used to simulate the future behavior of the system using iterative one-step ahead predictions. This results in stochastic distributions of the simulated states. We formulate the optimal control

problem in the stochastic MPC framework as proposed in [50].

$$\min_{\{\mathbf{u}_i\}} \mathbb{E}\left( l_f(\mathbf{x}_N) + \sum_{i=0}^{N-1} l_i(\mathbf{x}_i, \mathbf{u}_i) \right) \tag{3.2}$$

$$\text{s.t. } \mathbf{x}_{i+1} = \boldsymbol{h}(\mathbf{x}_i, \mathbf{u}_i) + \mathbf{B}_d \left( \boldsymbol{g}(\mathbf{x}_i, \mathbf{u}_i) + \boldsymbol{\epsilon}_i \right) \tag{3.3}$$

$$\mathbf{Pr}(\mathbf{x}_{i+1} \in \mathcal{X}_{i+1}) \geq p_x \tag{3.4}$$

$$\mathbf{Pr}(\mathbf{u}_i \in \mathcal{U}_i) \geq p_u \tag{3.5}$$

$$\mathbf{x}_0 = \mathbf{x}(k) \tag{3.6}$$

The optimization problem is solved over a sequence of control inputs. We use the concept of probabilistic reachable set as discussed in [52, 50] to reformulate the chance constraints deterministically.

### 3.2.1 Prediction Model and State Uncertainty Propagation

We apply structured kernel interpolation for online Gaussian process regression (WISKI) discussed in [72] to estimate the unknown function $\boldsymbol{g}$ from sequential observations of states. We consider that the measurements are corrupted with i.i.d. zero mean Gaussian noise with variance $\sigma_\epsilon^2$. Then we generate the regression targets from the nominal model and the noisy measurements of states using the following expression,

$$\mathbf{y}_k = \boldsymbol{g}(\mathbf{x}_k, \mathbf{u}_k) + \boldsymbol{\epsilon}_k = \mathbf{B}_d^\dagger \left( \mathbf{x}_{k+1} - \boldsymbol{h}(\mathbf{x}_k, \mathbf{u}_k) \right) \tag{3.7}$$

In the above equation $\mathbf{B}_d^\dagger$ represents the Moore-Penrose pseudo-inverse of $\mathbf{B}_d$. Often, we know the linear model ($\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$) of the dynamical system derived from the first principles. Using this we can rewrite our regression data generation

equation as,

$$\mathbf{y}_k = \mathbf{B}_d^{\dagger}\left(\mathbf{x}_{k+1} - \mathbf{A}\mathbf{x}_k - \mathbf{B}\mathbf{u}_k\right) \tag{3.8}$$

$$= \mathbf{B}_d^{\dagger}\left(\mathbf{x}_{k+1} - \tilde{\mathbf{A}}\mathbf{z}_k\right) \tag{3.9}$$

where, $\tilde{\mathbf{A}} = [\mathbf{A} \quad \mathbf{B}]$, and $\mathbf{z}_i^{\intercal} = [\mathbf{x}_k^{\intercal} \quad \mathbf{u}_k^{\intercal}]^{\intercal}$. The input feature vector of the regression problem is the augmented state control vector $\mathbf{z}_k \in \mathbb{R}^{n_x+n_u}$ and at each time instant $k$ we have new measurements, $\mathcal{D}_k = \{\mathbf{z}_k, \mathbf{y}_k\}$. We use $\mathcal{D}_k$ to update our model at each time step $k$. This results in a stochastic distribution of the approximation of $\boldsymbol{g}$. We denote the Gaussian process approximate of unknown dynamics $\boldsymbol{g}$ from noisy measurements as $\mathbf{d}$. Being an affine transformation of Gaussian vector $\mathbf{d}_{k-1}$, $\mathbf{x}_k$ will also be a Gaussian vector. Then, $\mathbf{x}_k$ and $\mathbf{d}_k$ can be approximated as a jointly Gaussian distribution at each step $k$ as shown below.

$$\begin{bmatrix}\mathbf{x}_k \\ \mathbf{d}_k\end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_k^x \\ \boldsymbol{\mu}_k^{\boldsymbol{d}}\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_k^x & \boldsymbol{\Sigma}_k^{xd} \\ \boldsymbol{\Sigma}_k^{dx} & \boldsymbol{\Sigma}_k^d\end{bmatrix}\right) \tag{3.10}$$

The predicted mean $\boldsymbol{\mu}_{i+1}^x$ and the covariance $\boldsymbol{\Sigma}_{i+1}^x$ can then be computed by using approach similar to extended Kalman filtering. Derivations are shown in A.4.

$$\boldsymbol{\mu}_{k+1}^x = \boldsymbol{h}(\boldsymbol{\mu}_k^x, \mathbf{u}_k) + \mathbf{B}_d\boldsymbol{\mu}_k^d \tag{3.11}$$

$$= \begin{bmatrix}\mathbf{A} & \mathbf{B}_d\end{bmatrix}\boldsymbol{\mu}_k^x + \mathbf{B}\mathbf{u}_k \tag{3.12}$$

$$\boldsymbol{\Sigma}_{k+1}^x = \begin{bmatrix}\nabla\boldsymbol{h}(\boldsymbol{\mu}_k^x, \mathbf{u}_k) & \mathbf{B}_d\end{bmatrix}\boldsymbol{\Sigma}_k\begin{bmatrix}\nabla\boldsymbol{h}(\boldsymbol{\mu}_k^x, \mathbf{u}_k) & \mathbf{B}_d\end{bmatrix}^{\intercal} \tag{3.13}$$

$$= \begin{bmatrix}\mathbf{A} & \mathbf{B}_d\end{bmatrix}\begin{bmatrix}\boldsymbol{\Sigma}_k^x & \boldsymbol{\Sigma}_k^{xd} \\ \boldsymbol{\Sigma}_k^{dx} & \boldsymbol{\Sigma}_k^d\end{bmatrix}\begin{bmatrix}\mathbf{A}^{\intercal} \\ \mathbf{B}_d^{\intercal}\end{bmatrix} \tag{3.14}$$

The mean $\boldsymbol{\mu}_k^d$ and the covariance matrix $\boldsymbol{\Sigma}_k^d$ of GP approximation, $\mathbf{d}_k$ and the cross covariance between states, $\mathbf{x}_k$ and GP approximation, $\mathbf{d}_k$, $\boldsymbol{\Sigma}_k^{xd}$ can be computed at each time instant using linearization of posterior mean of the Gaussian process with respect to the mean of predicting input (section 2.6.4). The resulting equations can be written as,

$$\boldsymbol{\mu}_k^d = \boldsymbol{\mu}^d(\boldsymbol{\mu}_k^x, \mathbf{u}_k) \tag{3.15}$$

$$\boldsymbol{\Sigma}_k^{xd} = \boldsymbol{\Sigma}_k^x \left(\nabla_x \boldsymbol{\mu}^d(\boldsymbol{\mu}_k^x, \mathbf{u}_k)\right)^\intercal \tag{3.16}$$

$$\boldsymbol{\Sigma}_k^d = \boldsymbol{\Sigma}^{\boldsymbol{d}}(\boldsymbol{\mu}_{\boldsymbol{k}}^{\boldsymbol{x}}, \mathbf{u}_{\boldsymbol{k}}) + \nabla_x \boldsymbol{\mu}^d(\boldsymbol{\mu}_k^x, \mathbf{u}_k)\boldsymbol{\Sigma}_k^x \left(\nabla_x \boldsymbol{\mu}^d(\boldsymbol{\mu}_k^x, \mathbf{u}_k)\right)^\intercal \tag{3.17}$$

In the above discussion we consider that the control inputs are deterministic. The extension to stochastic case is straightforward and can be computed by finding the joint distribution of states, control and the GP approximation of unknown function.

### 3.2.2 Cost Function

In model predictive control, the most popular example of cost function function is a quadratic cost on states and control. With the appropriate choice of weight matrices $\mathbf{Q} \succeq 0$ and $\mathbf{R} \succ 0$ we can write the expression for quadratic cost as

$$l(\mathbf{x}_i - \mathbf{x}_i^r, \mathbf{u}_i - \mathbf{u}_i^r) = \|\mathbf{x}_i - \mathbf{x}_i^r\|_{\mathbf{Q}}^2 + \|\mathbf{u}_i - \mathbf{u}_i^r\|_{\mathbf{R}}^2 \tag{3.18}$$

where, $\mathbf{x}^r$ and $\mathbf{u}^r$ represents the reference signal. In our problem, the states are distributed as Gaussian. This makes the cost function stochastic in nature. There are two main approaches for selecting a cost function. One is to take certainty equivalence approach, and use a quadratic cost function evaluated at the mean of our state distribution. [53, 54] uses this approach where the cost function is given

by

$$l(\mathbf{x}_i - \mathbf{x}_i^r, \mathbf{u}_i - \mathbf{u}_i^r) = \|\boldsymbol{\mu}_i^x - \mathbf{x}_i^r\|_{\mathbf{Q}}^2 + \|\mathbf{u}_i - \mathbf{u}_i^r\|_{\mathbf{R}}^2 \tag{3.19}$$

The other approach is to make use of the first and second moments of the states and compute the expected value of the quadratic cost given in Eq.(3.18) with respect to the state distribution [50].

$$\mathbb{E}\left(l(\mathbf{x}_i - \mathbf{x}_i^r, \mathbf{u}_i - \mathbf{u}_i^r)\right) = \|\boldsymbol{\mu}_i^x - \mathbf{x}_i^r\|_{\mathbf{Q}}^2 + \|\mathbf{u}_i - \mathbf{u}_i^r\|_{\mathbf{R}}^2 + tr\left(\mathbf{Q}\Sigma_i^x\right) \tag{3.20}$$

The derivation of the expected quadratic cost function is given in A.2. We denote the evaluation of the expected quadratic cost in terms of mean and variance as $\mathbb{E}\left(l(\mathbf{x}_i - \mathbf{x}_i^r, \mathbf{u}_i - \mathbf{u}_i^r)\right) = c_i\left(\boldsymbol{\mu}_i^x - \mathbf{x}_i^r, \mathbf{u}_i - \mathbf{u}_i^r, \boldsymbol{\Sigma}_i^x\right)$. In a similar way, we can compute the terminal cost $c_N\left(\boldsymbol{\mu}_N^x - \mathbf{x}_N^r, \boldsymbol{\Sigma}_N^x\right)$.

### 3.2.3 Chance Constraints Formulation

In the problem formulation, we assumed that the system's dynamics are unknown or partially known. The unknown function is learned from the measurement data using the probabilistic Gaussian process regression model. This results in a stochastic belief about the state, which expresses the uncertainty in the system dynamics based on the evidence. Due to the stochastic character of the problem, constraint fulfilment in the traditional sense cannot be guaranteed. A naive approach will be the direct use of expected state prediction for constraint satisfaction ignoring the uncertainty in the prediction. A more sophisticated way of handling the constraints is the construction of confidence sets,

$$Pr(\mathbf{x} \in \mathcal{X}) \geq p \tag{3.21}$$

with the probability of constraint satisfaction $p$. This type of probabilistic representation of constraints is also known as chance constraint formulation.

In [50] probabilistic reachable sets are used to reformulate the chance constraints on state using constraint tightening based on error $\mathbf{e}_i^x = \boldsymbol{\mu}_i^x - \mathbf{x}_i$. A probabilistic $i$-step reachable set is defined in [52] as an extension of the concept of reachable sets to stochastic systems.

**Definition 3.1** (Probabilistic $i$-step Reachable Set)**.** A set $\mathcal{R}$ is said to be a probabilistic $i$-step reachable set ($i$-step *PRS*) of probability level $p$ if

$$Pr(\mathbf{e}_i \in \mathcal{R} | \mathbf{e}_0 = 0) \geq p \tag{3.22}$$

Given the $i$-step *PRS* $\mathcal{R}_i^x$ of probability level $p_x$ for the state error $\mathbf{e}_i^x$ tightened constraints on the state can be defined with respect to the mean $\boldsymbol{\mu}_i^x$.

$$\boldsymbol{\mu}_i^x \in \mathcal{Z}_i = \mathcal{X}_i \ominus \mathcal{R}_i^x \tag{3.23}$$

where $\ominus$ represents the Pontryagin set difference. The Pontryagin's set difference for two set $\mathcal{P}$ and $\mathcal{Q}$ is defined as, $\mathcal{P} \ominus \mathcal{Q} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x} + \mathbf{q} \in \mathcal{P}, \forall \mathbf{q} \in \mathcal{Q}\}$. With Gaussian state $\mathbf{x}_i$ the uncertainty in each step $i$ can be fully specified by the covariance matrices $\boldsymbol{\Sigma}_i^x$. Then $i$-step *PRS* sets can be computed as a function of these covariance matrices [52]. Here, we consider the constrained set $\mathcal{X}_i$ given by a single half-space constraint.

$$\mathcal{X}_i^{hs} = \left\{ \mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}_i^\intercal \mathbf{x} \leq b_i \right\} \qquad \mathbf{a}_i \in \mathbb{R}^n, b_i \in \mathbb{R}_+ \tag{3.24}$$

Then the $i$-step *PRS* can be computed as a function of $\Sigma_i^x$ using the quantile function $\phi^{-1}(p_x)$ of a standard Gaussian random variable at the probability of constraint

satisfaction, $p_x$.

$$\mathcal{R}^x\left(\Sigma_i^x\right) = \left\{ \mathbf{e} \in \mathbb{R}^n \middle| \mathbf{a}_i^\intercal \mathbf{e} \leq \phi^{-1}\left(p_x\right) \sqrt{\mathbf{a}_i^\intercal \boldsymbol{\Sigma}_i^x \mathbf{a}_i} \right\} \tag{3.25}$$

The tightened state constraint can be computed as,

$$\mathcal{Z}_i^{hs}\left(\Sigma_i^x\right) = \left\{ \mathbf{z} \in \mathbb{R}^n \middle| \mathbf{a}_i^\intercal \mathbf{z} \leq b - \phi^{-1}\left(p_x\right) \sqrt{\mathbf{a}_i^\intercal \boldsymbol{\Sigma}_i^x \mathbf{a}_i} \right\} \tag{3.26}$$

The tightening of the slab constraints can be derived in a similar way, $\mathcal{X}_i^{sl} = \left\{ \mathbf{x} \mid |\mathbf{a}_i^\intercal \mathbf{x}| \leq b_i \right\}$, $\mathbf{a}_i \in \mathbb{R}^n, b_i \in \mathbb{R}_+$

$$\mathcal{Z}i^{hs}\left(\Sigma_i^x\right) = \left\{ \mathbf{z} \in \mathbb{R}^n \middle| \mathbf{a}_i^\intercal \mathbf{z} \leq b_i - \phi^{-1}\left(\frac{p_x + 1}{2}\right) \sqrt{\mathbf{a}_i^\intercal \boldsymbol{\Sigma}_i^x \mathbf{a}_i} \right\} \tag{3.27}$$

A derivation of tightened state constraints is shown in A.3. For stochastic control input, we can similarly compute the tightened constraints using the $i$-step *PRS*.

## 3.3 Online Gaussian Process-based MPC

Using the approximations discussed above, the learning-based MPC problem can be reformulated as

$$\min_{\{\mathbf{u}_i\}} \left( c_f(\boldsymbol{\mu}_N^x - \mathbf{x}_N^r, \boldsymbol{\Sigma}_N^x) + \sum_{i=0}^{N-1} c_i(\boldsymbol{\mu}_i^x - \mathbf{x}_i^r, \mathbf{u}_i - \mathbf{u}_i^r, \boldsymbol{\Sigma}_i^x) \right) \tag{3.28}$$

$$\text{s.t. } \boldsymbol{\mu}_{i+1}^x = \boldsymbol{h}(\boldsymbol{\mu}_i^x, \mathbf{u}_i) + \mathbf{B}_d \boldsymbol{\mu}^d(\boldsymbol{\mu}_i^x, \mathbf{u}_i) \tag{3.29}$$

$$\boldsymbol{\Sigma}_{i+1}^x = [\nabla \boldsymbol{h}(\boldsymbol{\mu}_N^x, \boldsymbol{\Sigma}_N^x) \;\; \mathbf{B}_d] \, \boldsymbol{\Sigma}_i \, [\nabla \boldsymbol{h}(\boldsymbol{\mu}_N^x, \boldsymbol{\Sigma}_N^x) \;\; \mathbf{B}_d]^T \tag{3.30}$$

$$\boldsymbol{\mu}_{i+1}^x \in \mathcal{Z}^{hs}\left(\boldsymbol{\Sigma}_{i+1}^x\right) \tag{3.31}$$

$$\mathbf{u}_i \in \mathcal{U} \tag{3.32}$$

$$\boldsymbol{\mu}_0^x = \mathbf{x}(k), \quad \boldsymbol{\Sigma}_{\mathbf{0}}^{\boldsymbol{x}} = 0 \tag{3.33}$$

for $i = 0, 1, ..., N-1$. At each instant of time, the solution of the optimal control problem results in a sequence of optimal control $\mathbf{u}_0^\star, ...., \mathbf{u}_{N-1}^\star$. The MPC control law is obtained in a receding fashion by applying the first element $\mathbf{u}_0^\star$ of the control sequence. The above non-linear MPC formulation results in a non-convex optimization problem due to the computation of the predictive second moment of the state distribution and finding a global optima is not guaranteed. We use Sequential Quadratic Programming to solve this problem. Also, at every optimization step the function evaluation depends on the prediction of GP. This makes solving the optimization problem time-consuming. We applied our approach to the pendulum problem. In the next chapter, we will show our results.

# Chapter 4

# Simulation Results

## 4.1 Pendulum Problem

We consider the problem of simple pendulum control as shown in Figure 4.1. The aim is to position the pendulum arm vertically upward starting from a known initial position. The system dynamics is described by a non-linear continuous-time model of the system (for simulation purposes) [32]. The states and the control input of the system are the angular position (rad) $x_1$, the angular velocity (rad/sec) $x_2$ and the torque (N-m) applied to the system $u$. We consider that the approximate linear model of the system is known. Euler's method is used to discretize the linear part of the system dynamics with sampling time, $T_s = 50$ ms. We assume that the non-linearity only affects the angular velocity and we are interested in learning this only. Then we can write, $\mathbf{B}_d = [0 \quad 1]^\intercal$. Then the system dynamics becomes the following.

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{B}_d \boldsymbol{g}(\mathbf{x}_k, \mathbf{u}_k) \tag{4.1}$$

Matrix $\mathbf{A}$ and matrix $\mathbf{B}$ can be found in [32]. For simulating the true system $\boldsymbol{g}$ is

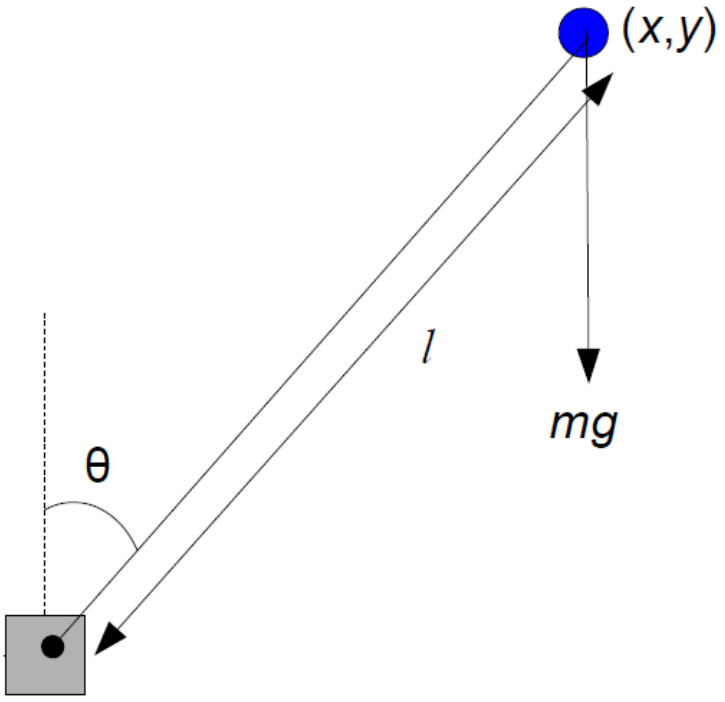


FIGURE 4.1: Pendulum System

computed using the values given in [32]. We use kernel interpolation-based online Gaussian process regression discussed in section 2.8 to learn the unknown function $\boldsymbol{g}$. Initially, we do not have any data point. As time progresses we collect data and the update the model at every time step. We considered the squared exponential kernel, and the hyperparameters of the kernel are learned at every step. Initially, the inducing points are taken on a three dimensional grid of $0.01 \times 0.01 \times 0.01$. The signal variance and the length-scale parameters are initialized randomly at small values.

We considered quadratic cost function with $\mathbf{Q} = diag([5.0, 0.1]^\intercal)$ and $\mathbf{R} = 0.01$. The prediction horizon is chosen to be $N = 12$. We consider two cases for our problem, one without any constraints on the system and the other with constraint on control input $(-3 \leq u \leq 3)$. For every case, we consider three scenarios for future state prediction, (i) using only the nominal model, (ii) using the sparse GP

model learned offline with the nominal model, and (iii) using the proposed approach of online learning with the nominal model. The results are shown in Figure (4.2)-Figure (4.5). In every figure, the left plot shows the results without any constraints on the system and case two shows the results with control constraints.

Figure 4.2 shows the angular position of the pendulum arm. The learning-based approaches, GPMPC with offline learning (yellow), and the proposed approach (green) outperform the nominal model-based MPC which is not surprising. Initially, the offline GPMPC performs well as compared to the proposed approach due to less uncertainty in the learned model. The proposed approach starts with an empty data set; thus,the uncertainty in the model remains high at the initial stage of learning. As we collect more, the uncertainty reduces and the proposed algorithm works well, eventually outperforming the offline GPMPC approach after some time. Similar results can be seen for angular velocity also as shown in Figure 4.3. The plots for control input is shown in Figure 4.4. In Figure 4.5, the plots for the MPC cost function are shown. For offline GPMPC and nominal MPC we can see a monotonic decrease in the cost function, but for the proposed approach it is not true due to the initial uncertainty in the model.

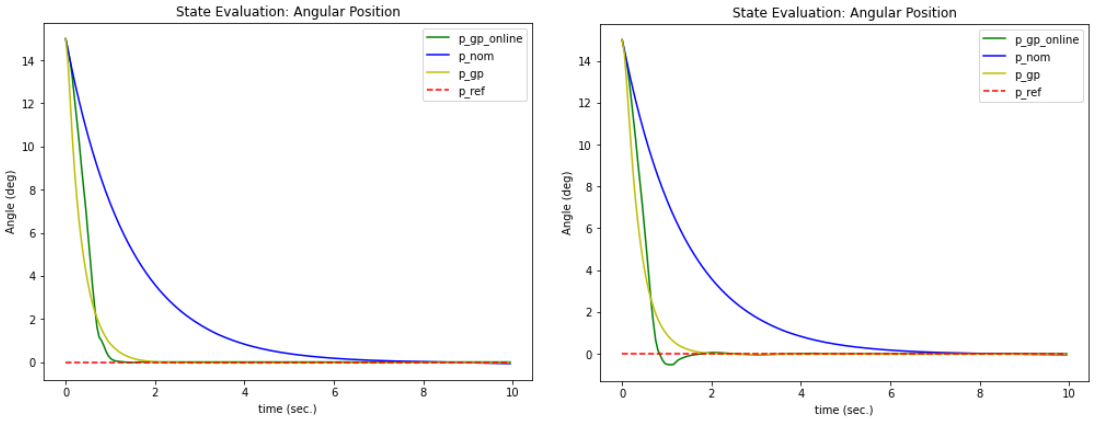


FIGURE 4.2: State trajectory, angular position (degree); left plot: unconstrained; right plot: constraint on input.The red dotted line represents the reference signal (p_ref). The blue line represents the result corresponding to the nominal model based MPC (p_nom). The results for GPMPC with offline learning (p_gp) and the proposed approach (p_gp_online) is shown by the yellow and green line respectively.

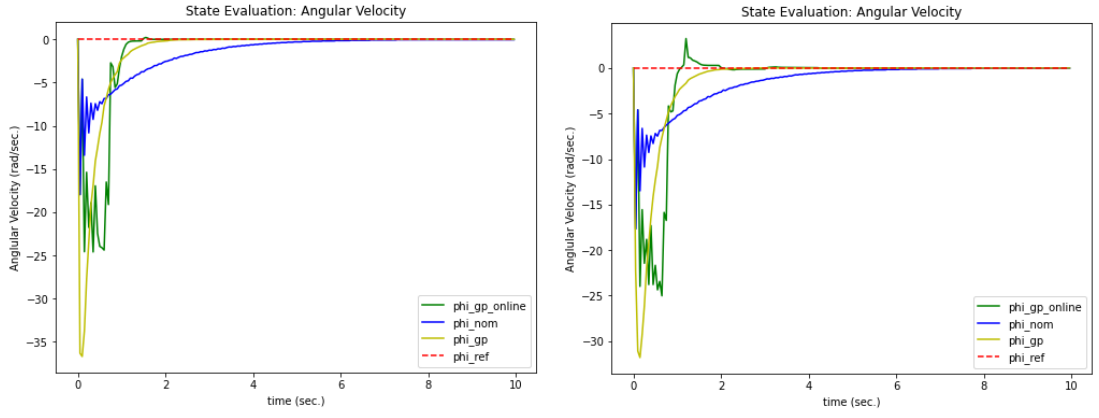


FIGURE 4.3: State trajectory, angular velocity (degree/sec.); left plot: unconstrained; right plot: constraint on input

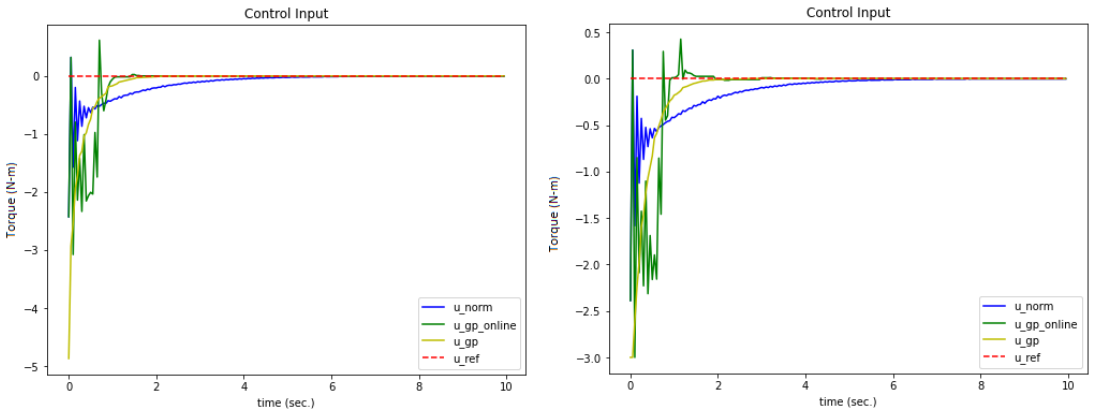


FIGURE 4.4: Control input, torque (Nm); left plot: unconstrained; right plot: constraint on input

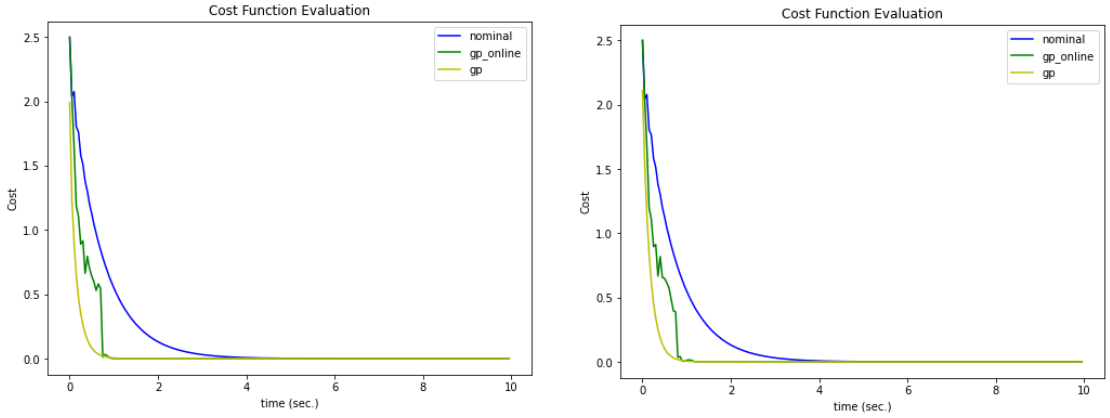


FIGURE 4.5: Evaluation of the cost function; left plot: unconstrained; right plot: constraint on input

# Chapter 5

# Conclusion and Scope for Future Work

This dissertation focuses on developing an approach that solves the optimal control problem to achieve certain performance measure corresponding to a system with partially known dynamics. This work comprises two parts. The first part includes the identification of the unknown part of system dynamics from measurement data. The concept of Bayesian supervised learning is used to learn the unknown part of the transition function and quantify the uncertainty within the learned model. The kernel interpolation-based method is used to overcome the difficulties of Gaussian process regression with streaming data. In the second part, the model predictive control technique is used to solve the optimal control problem that has the ability to handle the system constraints ensuring the safety of the system. The learned dynamics together with the nominal model is used to simulate the future behaviour of the system. The uncertainty in the learned model is used for planning and chance constraint formulation. The proposed approach achieves better performance than the Gaussian process model predictive control approach with offline learning.

## 5.1 Future Work

Though the online learning based model predictive control with Gaussian processes approach works well in the simulated environment, several open problem exists for this approach.

### 5.1.1 Mathematical Validation

Gaussian processes with inducing point approximation generally works as black box modelling. In this approach the information contained in the data set is summarized by a set of pseudo input points. Showing mathematical analysis of recursive feasibility and stability of the proposed MPC scheme is difficult. Future work lies in developing a mathematical framework for the proposed approach ensuring stability and recursive feasibility.

### 5.1.2 Experimental Validation

The main motivation behind online learning of the unknown system is that it becomes capable of working with a changing environment. Also, learning the unknown dynamics of the system with Gaussian process model results in a nonlinear MPC problem making its application difficult for a fast moving system. Though we have applied the proposed approach in simulated environment to a simple pendulum problem, a successful test of the algorithm to a practical system will work as a proof of concept.

# Appendix A

# DERIVATIONS

## A.1 Linearization of Posterior Mean Function

In linearization of the posterior mean function method [42], the posterior mean function is linearized using $1^{st}$ order Taylor series expansion around $\boldsymbol{\mu}^\star$ to approximate the mean of marginal predictive distribution.

$$\begin{aligned}\mathbb{E}[f^\star|\mathbf{y},\mathbf{X}] &= \mathbb{E}_{\mathbf{x}^\star\sim p(\mathbf{x}^\star)}\left[\bar{\mathbf{m}}(\mathbf{x}^\star)\right] \\ &\approx \mathbb{E}\left[\bar{\mathbf{m}}(\boldsymbol{\mu}^\star) + (\nabla_{\mathbf{x}^\star}\bar{\mathbf{m}}(\boldsymbol{\mu}^\star))^\intercal\left(\mathbf{x}^\star - \boldsymbol{\mu}^\star\right)\right] + \mathcal{O}(2) \\ &= \bar{\mathbf{m}}(\boldsymbol{\mu}^\star)\end{aligned} \tag{A.1}$$

Marginal predictive variance is computed similarly,

$$\begin{aligned}\mathbb{V}[f^\star|\mathbf{y},\mathbf{X}] &= \mathbb{V}_{\mathbf{x}^\star\sim p(\mathbf{x}^\star)}\left[\bar{\mathbf{m}}(\mathbf{x}^\star)\right] + \mathbb{E}_{\mathbf{x}^\star\sim p(\mathbf{x}^\star)}\left[\bar{\mathbf{K}}(\mathbf{x}^\star)\right] \\ &= \nabla_{\mathbf{x}^\star}\bar{m}(\boldsymbol{\mu}^\star)\boldsymbol{\Sigma}^\star\nabla_{\mathbf{x}^\star}\bar{m}(\boldsymbol{\mu}^\star)^\intercal + \bar{\mathbf{K}}(\boldsymbol{\mu}^\star)\end{aligned} \tag{A.2}$$

The first term can be computed as,

$$\begin{aligned}\mathbb{V}\left[\bar{\mathbf{m}}(\mathbf{x}^\star)\right] &= \mathbb{V}\left[\bar{\mathbf{m}}(\boldsymbol{\mu}^\star) + \nabla_{\mathbf{x}^\star}\bar{\mathbf{m}}(\boldsymbol{\mu}^\star)\left(\mathbf{x}^\star - \boldsymbol{\mu}^\star\right)\right] \\ &= \mathbb{V}\left[\nabla_{\mathbf{x}^\star}\bar{\mathbf{m}}(\boldsymbol{\mu}^\star)\left(\mathbf{x}^\star - \boldsymbol{\mu}^\star\right)\right] \\ &= \nabla_{\mathbf{x}^\star}\bar{m}(\boldsymbol{\mu}^\star)\mathbb{V}\left[\left(\mathbf{x}^\star - \boldsymbol{\mu}^\star\right)\right]\nabla_{\mathbf{x}^\star}\bar{m}(\boldsymbol{\mu}^\star)^\intercal \\ &= \nabla_{\mathbf{x}^\star}\bar{m}(\boldsymbol{\mu}^\star)\boldsymbol{\Sigma}^\star\nabla_{\mathbf{x}^\star}\bar{m}(\boldsymbol{\mu}^\star)^\intercal\end{aligned} \tag{A.3}$$

The second term is be computed using Taylor series expansion of $\bar{\mathbf{K}}$ around $\boldsymbol{\mu}^\star$

$$\begin{aligned}\mathbb{E}\left[\bar{\mathbf{K}}(\mathbf{x}^\star)\right] &= \mathbb{E}\left[\bar{\mathbf{K}}(\boldsymbol{\mu}^\star) + \left(\nabla_{\mathbf{x}^\star}\bar{\mathbf{K}}(\boldsymbol{\mu}^\star)\right)^\intercal\left(\mathbf{x}^\star - \boldsymbol{\mu}^\star\right)\right] \\ &= \bar{\mathbf{K}}(\boldsymbol{\mu}^\star)\end{aligned} \tag{A.4}$$

Then the mean of the predictive distribution at test input $\mathbf{x}^\star$ becomes,

$$\begin{aligned}\bar{\mathbf{m}} &= \mathbf{k}\left(\mathbf{x}^\star, \mathbf{X}\right)\left(\mathbf{K}\left(\mathbf{X}, \mathbf{X}\right) + \sigma_\epsilon^2 I\right)^{-1}\mathbf{y} \\ &= \mathbf{k}\left(\mathbf{x}^\star, \mathbf{X}\right)\boldsymbol{\alpha}\end{aligned} \tag{A.5}$$

In the above expression, $\boldsymbol{\alpha} = \left(\mathbf{K}\left(\mathbf{X}, \mathbf{X}\right) + \sigma_\epsilon^2 I\right)^{-1}\mathbf{y}$, is independent of $\mathbf{x}^\star$, therefore to calculate the derivative of the posterior mean function we only have to differentiate the kernel. For the squared exponential covariance function, the derivative of the kernel between $\mathbf{x}^\star$ and a training point $\mathbf{x}_i$ is

$$\begin{aligned}\frac{\partial k(\mathbf{x}^\star, \mathbf{x}_i)}{\partial \mathbf{x}^\star} &= \frac{\partial}{\partial \mathbf{x}^\star}\left\{\sigma_{SE}^2 \exp\left(-\frac{1}{2}(\mathbf{x}^\star - \mathbf{x}_i)^\intercal\Lambda^{-1}(\mathbf{x}^\star - \mathbf{x}_i)\right)\right\} \\ &= k(\mathbf{x}^\star, \mathbf{x}_i)\frac{\partial}{\partial \mathbf{x}^\star}\left\{-\frac{1}{2}(\mathbf{x}^\star - \mathbf{x}_i)^\intercal\Lambda^{-1}(\mathbf{x}^\star - \mathbf{x}_i)\right\} \\ &= \Lambda^{-1}(\mathbf{x}^\star - \mathbf{x}_i)k(\mathbf{x}^\star, \mathbf{x}_i)\end{aligned} \tag{A.6}$$

which is a $D \times 1$ vector. Differentiating (A.5) with respect to $\mathbf{x}^\star$ and substituting (A.6) we get the expression of the derivative of the posterior mean function

$$\begin{aligned} \frac{\partial \bar{\mathbf{m}}}{\partial \mathbf{x}^\star} &= \frac{\partial \mathbf{k}(\mathbf{x}^\star, \mathbf{X})}{\partial \mathbf{X}} \boldsymbol{\alpha} \\ &= -\boldsymbol{\Lambda}^{-1} \tilde{\mathbf{X}}^\star \left( \mathbf{k}(\mathbf{x}^\star, \mathbf{X})^T \odot \boldsymbol{\alpha} \right) \end{aligned} \tag{A.7}$$

where, $\tilde{\mathbf{X}}^\star = [\mathbf{x}^\star - \mathbf{x}_1, .....\mathbf{x}^\star - \mathbf{x}_N]$ is a $D \times N$ matrix. The derivative $\frac{\partial \bar{\mathbf{m}}}{\partial \mathbf{x}^\star}$ in (A.7) is a vector of $D \times 1$ dimension. $\odot$ in the above expression represents the element-wise product.

## A.2 MPC Cost Function

The expression of the expected quadratic stage cost can be derived as follows.

$$\begin{aligned} \mathbb{E}\left[ l\left( \mathbf{x}_i - \mathbf{x}^r, \mathbf{u}_i - \mathbf{u}^r \right) \right] &= \mathbb{E}\left[ \left( \mathbf{x}_i - \mathbf{x}^r \right)^T \mathbf{Q} \left( \mathbf{x}_i - \mathbf{x}^r \right) + \left( \mathbf{u}_i - \mathbf{u}^r \right)^T \mathbf{R} \left( \mathbf{u}_i - \mathbf{u}^r \right) \right] \\ &= \int \int \left[ \left( \mathbf{x}_i - \mathbf{x}^r \right)^T \mathbf{Q} \left( \mathbf{x}_i - \mathbf{x}^r \right) + \left( \mathbf{u}_i - \mathbf{u}^r \right)^T \mathbf{R} \left( \mathbf{u}_i - \mathbf{u}^r \right) \right] p(\mathbf{x}, \mathbf{u}) d\mathbf{x} d\mathbf{u} \\ &= \int \int \left[ \left( \mathbf{x}_i - \mathbf{x}^r \right)^T \mathbf{Q} \left( \mathbf{x}_i - \mathbf{x}^r \right) \right] p(\mathbf{x}, \mathbf{u}) d\mathbf{x} d\mathbf{u} \\ &+ \int \int \left[ \left( \mathbf{u}_i - \mathbf{u}^r \right)^T \mathbf{R} \left( \mathbf{u}_i - \mathbf{u}^r \right) \right] p(\mathbf{x}, \mathbf{u}) d\mathbf{x} d\mathbf{u} \end{aligned} \tag{A.8}$$

In (A.8) we will derive the first expression of right-hand side. The other expression can be derived in a similar manner.

$$\int \int \left[ \left( \mathbf{x}_i - \mathbf{x}^r \right)^T \mathbf{Q} \left( \mathbf{x}_i - \mathbf{x}^r \right) \right] p(\mathbf{x}, \mathbf{u}) d\mathbf{x} d\mathbf{u} \tag{A.9}$$

$$
\begin{aligned}
&= \int\int \left[ (\mathbf{x}_i - \mathbf{x}^r)^T \mathbf{Q} (\mathbf{x}_i - \mathbf{x}^r) \right] p(\mathbf{u}|\mathbf{x}) p(\mathbf{x}) d\mathbf{x} d\mathbf{u} \\
&= \int \left[ (\mathbf{x}_i - \mathbf{x}^r)^T \mathbf{Q} (\mathbf{x}_i - \mathbf{x}^r) \right] p(\mathbf{x}) d\mathbf{x} \\
&= \mathbf{x}^{r\intercal} \mathbf{Q} \mathbf{x}^r - 2 \int \mathbf{x}^{r\intercal} \mathbf{Q} \mathbf{x}_i p(\mathbf{x}) d\mathbf{x} + \int \mathbf{x}_i{}^{\intercal} \mathbf{Q} \mathbf{x}_i p(\mathbf{x}) d\mathbf{x} \\
&= \mathbf{x}^{r\intercal} \mathbf{Q} \mathbf{x}^r - 2\mathbf{x}^{r\intercal} \mathbf{Q} \boldsymbol{\mu}_i^x + \boldsymbol{\mu}_i^{x\intercal} + \int \left( \mathbf{x}_i{}^{\intercal} \mathbf{Q} \mathbf{x}_i - \boldsymbol{\mu}_i^{x\intercal} \mathbf{Q} \boldsymbol{\mu}_i^x \right) p(\mathbf{x}) d\mathbf{x} \\
&= \left[ (\boldsymbol{\mu}_i^x - \mathbf{x}^r)^T \mathbf{Q} (\boldsymbol{\mu}_i^x - \mathbf{x}^r) \right] + \int tr \left( \mathbf{Q} \left( \mathbf{x}_i \mathbf{x}_i^{\intercal} - \boldsymbol{\mu}_i^x \boldsymbol{\mu}_i^{x\intercal} \right) \right) \\
&= \left[ (\boldsymbol{\mu}_i^x - \mathbf{x}^r)^T \mathbf{Q} (\boldsymbol{\mu}_i^x - \mathbf{x}^r) \right] + tr \left( \mathbf{Q} \boldsymbol{\Sigma}_{\boldsymbol{i}}^{\boldsymbol{x}} \right)
\end{aligned}
$$

Similarly, we can derive the expression for the terminal cost.

## A.3 Chance Constraint Formulation

The tightened state and control input constraints with respect to their mean can be written as

$$\boldsymbol{\mu}_i^x \in \mathcal{Z}_i = \mathcal{X}_i \ominus \mathcal{R}_i^x \tag{A.10}$$

$$\boldsymbol{\mu}_i^u \in \mathcal{V}_i = \mathcal{U}_i \ominus \mathcal{R}_i^u \tag{A.11}$$

From Eq. (A.10) we can say that,

$$\boldsymbol{\mu}_i^x \in \mathcal{Z}_i \Rightarrow Pr(\mathbf{x}_i = \boldsymbol{\mu}_i^x + \mathbf{e}_i^x \in \mathcal{X}) \geq Pr(\mathbf{e}_i^x \in \mathcal{R}^x) \geq p_x$$

Here we only consider the half-space constraints given by,

$$\mathcal{X}_i^{hs} = \left\{ \mathbf{x} \mid \mathbf{h}_i^{\intercal} \mathbf{x} \leq b_i \right\} \qquad \mathbf{h}_i \in \mathbb{R}^n, b_i \in \mathbb{R}_+ \tag{A.12}$$

We are interested in finding the marginal distribution of the error in the direction of half space, $\mathbf{h}_i^\intercal \mathbf{e}_i^x$. With Gaussian state distribution the error distribution will also be a Gaussian with mean and variance computed as below,

$$\begin{aligned}\mathbb{E}\left[\mathbf{h}_i^\intercal \mathbf{e}_i^x\right] &= \mathbb{E}\left[\mathbf{h}_i^\intercal \left(\boldsymbol{\mu}_i^x - \mathbf{x}_i\right)\right] \\ &= 0\end{aligned}$$

$$\begin{aligned}\mathbb{V}\left[\mathbf{h}_i^\intercal \mathbf{e}_i^x\right] &= \mathbb{E}\left[(\mathbf{h}_i^\intercal \mathbf{e}_i^x)(\mathbf{h}_i^\intercal \mathbf{e}_i^x)^\intercal\right] \\ &= \mathbf{h}_i^\intercal \mathbb{E}\left[(\mathbf{e}_i^x)(\mathbf{e}_i^x)^\intercal\right] \mathbf{h}_i \\ &= \mathbf{h}_i^\intercal \mathbb{E}\left[(\boldsymbol{\mu}_i^x - \mathbf{x}_i)(\boldsymbol{\mu}_i^x - \mathbf{x}_i)^\intercal\right] \mathbf{h}_i \\ &= \mathbf{h}_i^\intercal \boldsymbol{\Sigma}_i^x \mathbf{h}_i\end{aligned}$$

Using the quantile function of a standard Gaussian random variable $\phi^{-1}$ at the probability of constraint satisfaction $p_x$, $i$-step *PRS* can be written as

$$\mathcal{R}^x\left(\Sigma_i^x\right) = \left\{ \mathbf{e} \middle| \mathbf{h}_i^\intercal \mathbf{e} \leq \phi^{-1}\left(p_x\right) \sqrt{\mathbf{h}_i^\intercal \boldsymbol{\Sigma}_i^x \mathbf{h}_i} \right\} \tag{A.13}$$

## A.4 Uncertainty Propagation

The state update equation can be written as,

$$\mathbf{x}_{k+1} = \mathbf{h}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{B}_d \mathbf{d} \tag{A.14}$$

At $k = 0$, $\mathbf{x}_0$ is deterministic (known from the measurement). Then, $\mathbf{x}_1$ becomes Gaussian.

$$\mathbf{x}_1 = \mathbf{h}(\mathbf{x}_0, \mathbf{u}_0) + \mathbf{B}_d \boldsymbol{\mu}_0^d \tag{A.15}$$

For the next states, we use linearization of posterior mean approach to predict at a Gaussian state. Similarly we can linearize $\mathbf{h}$ around $\boldsymbol{\mu}_k^x$ to find our new update equation for mean,

$$\boldsymbol{\mu}_{k+1}^x = \mathbf{h}(\boldsymbol{\mu}_k^x, \mathbf{u}_k) + \mathbf{B}_d \boldsymbol{\mu}_k^d \tag{A.16}$$

For variance updation we follow similar steps, that gives

$$\begin{aligned}
\boldsymbol{\Sigma}_{k+1}^x &= \mathbb{V}\left[\mathbf{h}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{B}_d \mathbf{d}\right] \\
&= \mathbb{V}\left[\mathbf{h}(\boldsymbol{\mu}_k^x, \mathbf{u}_k) + \nabla_x \mathbf{h}(\boldsymbol{\mu}_k^x, \mathbf{u}_k)(\mathbf{x}_k - \boldsymbol{\mu}_k^x) + \mathbf{B}_d \mathbf{d}\right] \\
&= \nabla_x \mathbf{h}(\boldsymbol{\mu}_k^x, \mathbf{u}_k) \boldsymbol{\Sigma}_k^x \left(\nabla_x \mathbf{h}(\boldsymbol{\mu}_k^x, \mathbf{u}_k)\right)^\intercal \\
&+ \nabla_x \mathbf{h}(\boldsymbol{\mu}_k^x, \mathbf{u}_k) \boldsymbol{\Sigma}_k^{xd} \mathbf{B}_d^\intercal + \mathbf{B}_d \boldsymbol{\Sigma}_k^{dx} \left(\nabla_x \mathbf{h}(\boldsymbol{\mu}_k^x, \mathbf{u}_k)\right)^\intercal + \mathbf{B}_d \boldsymbol{\Sigma}_k^d \mathbf{B}_d^\intercal \\
&= \begin{bmatrix} \nabla_x \mathbf{h}(\boldsymbol{\mu}_k^x, \mathbf{u}_k) & \mathbf{B}_d \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_k^x & \boldsymbol{\Sigma}_k^{xd} \\ \boldsymbol{\Sigma}_k^{dx} & \boldsymbol{\Sigma}_k^d \end{bmatrix} \begin{bmatrix} \nabla_x \mathbf{h}(\boldsymbol{\mu}_k^x, \mathbf{u}_k)^\intercal \\ \mathbf{B}_d^\intercal \end{bmatrix} \\
&= \begin{bmatrix} \nabla_x \mathbf{h}(\boldsymbol{\mu}_k^x, \mathbf{u}_k) & \mathbf{B}_d \end{bmatrix} \boldsymbol{\Sigma}_k \begin{bmatrix} \nabla_x \mathbf{h}(\boldsymbol{\mu}_k^x, \mathbf{u}_k) & \mathbf{B}_d \end{bmatrix}^\intercal
\end{aligned} \tag{A.17}$$

# References


[1] Donald E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2004.

[2] I Michael Ross and Mark Karpenko. A review of pseudospectral optimal control: From theory to flight. *Annual Reviews in Control*, 36(2):182–197, 2012.

[3] James E Bobrow, Steven Dubowsky, and John S Gibson. Time-optimal control of robotic manipulators along specified paths. *The international journal of robotics research*, 4(3):3–17, 1985.

[4] Robert Dorfman. An economic interpretation of optimal control theory. *The American Economic Review*, 59(5):817–831, 1969.

[5] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436–444, 2015.

[7] Erik Cambria and Bebo White. Jumping nlp curves: A review of natural language processing research. *IEEE Computational intelligence magazine*, 9(2): 48–57, 2014.

[8] Nesreen K Ahmed, Amir F Atiya, Neamat El Gayar, and Hisham El-Shishiny. An empirical comparison of machine learning models for time series forecasting. *Econometric reviews*, 29(5-6):594–621, 2010.

[9] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[10] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[11] Hoseinali Borhan, Ardalan Vahidi, Anthony M Phillips, Ming L Kuang, Ilya V Kolmanovsky, and Stefano Di Cairano. Mpc-based energy management of a power-split hybrid electric vehicle. *IEEE Transactions on Control Systems Technology*, 20(3):593–603, 2011.

[12] Gianni Bianchini, Marco Casini, Daniele Pepe, Antonio Vicino, and Giovanni Gino Zanvettor. An integrated mpc approach for demand-response heating and energy storage operation in smart buildings. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 3865–3870. IEEE, 2017.

[13] Xianzhong Chen, Mohsen Heidarinejad, Jinfeng Liu, and Panagiotis D Christofides. Distributed economic mpc: Application to a nonlinear chemical process network. *Journal of Process Control*, 22(4):689–699, 2012.

[14] B Holenda, E Domokos, A Redey, and J Fazakas. Dissolved oxygen control of the activated sludge wastewater treatment process using model predictive control. *Computers & Chemical Engineering*, 32(6):1270–1278, 2008.

[15] Arne Linder and Ralph Kennel. Model predictive control for electrical drives. In *2005 IEEE 36th Power Electronics Specialists Conference*, pages 1793–1799. IEEE, 2005.

[16] Saverio Bolognani, Silverio Bolognani, Luca Peretti, and Mauro Zigliotto. Design and implementation of model predictive control for electrical motor drives. *IEEE Transactions on industrial electronics*, 56(6):1925–1936, 2008.

[17] Piotr J Serkies and Krzysztof Szabat. Application of the mpc to the position control of the two-mass drive system. *IEEE Transactions on Industrial Electronics*, 60(9):3679–3688, 2012.

[18] MM Kale and AJ Chipperfield. Stabilized mpc formulations for robust reconfigurable flight control. *Control Engineering Practice*, 13(6):771–788, 2005.

[19] David Q Mayne, James B Rawlings, Christopher V Rao, and Pierre OM Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[20] Robert R Bitmead, Michel Gevers, and Vincent Wertz. Adaptive optimal control the thinking man's gpc. 1990.

[21] R Berber. Control of batch reactors-a review (reprinted from methods of model based process control, 1995). *Chemical engineering research & design*, 74(1): 3–20, 1996.

[22] James B Rawlings and Kenneth R Muske. The stability of constrained receding horizon control. *IEEE transactions on automatic control*, 38(10):1512–1516, 1993.

[23] Alberto Bemporad and Manfred Morari. Robust model predictive control: A survey. In *Robustness in identification and control*, pages 207–226. Springer, 1999.

[24] Pierre OM Scokaert and James B Rawlings. Constrained linear quadratic regulation. *IEEE Transactions on automatic control*, 43(8):1163–1169, 1998.

[25] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[26] Shankar Sastry, Marc Bodson, and James F Bartram. Adaptive control: stability, convergence, and robustness, 1990.

[27] Frank L Lewis, Draguna Vrabie, and Kyriakos G Vamvoudakis. Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers. *IEEE Control Systems Magazine*, 32(6):76–105, 2012.

[28] Sinan Çalışır and Meltem Kurt Pehlivanoğlu. Model-free reinforcement learning algorithms: A survey. In *2019 27th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE, 2019.

[29] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 2013. URL `http://arxiv.org/abs/1312.5602`. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

[30] Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020.

[31] Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? *Structural safety*, 31(2):105–112, 2009.

[32] Marc Peter Deisenroth. *Efficient reinforcement learning using Gaussian processes*, volume 9. KIT Scientific Publishing, 2010.

[33] Kenneth J Hunt, D Sbarbaro, R Żbikowski, and Peter J Gawthrop. Neural networks for control systems—a survey. *Automatica*, 28(6):1083–1112, 1992.

[34] Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: a survey. *Cognitive processing*, 12(4):319–340, 2011.

[35] Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

[36] Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning for control. *Lazy learning*, pages 75–113, 1997.

[37] John G Kuschewski, Stefen Hui, and Stanislaw H Zak. Application of feedforward neural networks to dynamical system identification and control. *IEEE Transactions on Control Systems Technology*, 1(1):37–49, 1993.

[38] Gianluigi Pillonetto and Giuseppe De Nicolao. A new kernel-based approach for linear system identification. *Automatica*, 46(1):81–93, 2010.

[39] Girish Chowdhary, Hassan A Kingravi, Jonathan P How, and Patricio A Vela. Bayesian nonparametric adaptive control using gaussian processes. *IEEE transactions on neural networks and learning systems*, 26(3):537–550, 2014.

[40] Joaquin Quinonero Candela, Agathe Girard, Jan Larsen, and Carl Edward Rasmussen. Propagation of uncertainty in bayesian kernel models-application to multiple-step ahead forecasting. In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).*, volume 2, pages II–701. IEEE, 2003.

[41] J Quinonero Candela. Learning with uncertainty-gaussian processes and relevance vector machines. *Technical University of Denmark, Copenhagen*, 2004.

[42] Agathe Girard, Carl Rasmussen, Joaquin Q Candela, and Roderick Murray-Smith. Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting. *Advances in neural information processing systems*, 15, 2002.

[43] Agathe Girard, Carl Edward Rasmussen, J Quinonero-Candela, R Murray-Smith, O Winther, and J Larsen. Multiple-step ahead prediction for non linear dynamic systems–a gaussian process treatment with propagation of the uncertainty. *Advances in neural information processing systems*, 15:529–536, 2002.

[44] Marc Peter Deisenroth, Carl Edward Rasmussen, and Dieter Fox. Learning to control a low-cost manipulator using data-efficient reinforcement learning. *Robotics: Science and Systems VII*, 7:57–64, 2011.

[45] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

[46] Juš Kocijan, Roderick Murray-Smith, Carl Edward Rasmussen, and Agathe Girard. Gaussian process model based predictive control. In *Proceedings of the 2004 American control conference*, volume 3, pages 2214–2219. IEEE, 2004.

[47] Edgar D Klenske, Melanie N Zeilinger, Bernhard Schölkopf, and Philipp Hennig. Gaussian process-based predictive control for periodic error correction. *IEEE Transactions on Control Systems Technology*, 24(1):110–121, 2015.

[48] Chris J Ostafew, Angela P Schoellig, Timothy D Barfoot, and Jack Collier. Learning-based nonlinear model predictive control to improve vision-based mobile robot path tracking. *Journal of Field Robotics*, 33(1):133–152, 2016.

[49] Sanket Kamthe and Marc Deisenroth. Data-efficient reinforcement learning with probabilistic model predictive control. In *International conference on artificial intelligence and statistics*, pages 1701–1710. PMLR, 2018.

[50] Lukas Hewing, Alexander Liniger, and Melanie N Zeilinger. Cautious nmpc with gaussian process dynamics for autonomous miniature race cars. In *2018 European Control Conference (ECC)*, pages 1341–1348. IEEE, 2018.

[51] Lukas Hewing, Juraj Kabzan, and Melanie N Zeilinger. Cautious model predictive control using gaussian process regression. *IEEE Transactions on Control Systems Technology*, 28(6):2736–2743, 2019.

[52] Lukas Hewing and Melanie N Zeilinger. Stochastic model predictive control for linear systems using probabilistic reachable sets. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 5182–5188. IEEE, 2018.

[53] Guillem Torrente, Elia Kaufmann, Philipp Föhn, and Davide Scaramuzza. Data-driven mpc for quadrotors. *IEEE Robotics and Automation Letters*, 6 (2):3769–3776, 2021.

[54] Andrea Carron, Elena Arcari, Martin Wermelinger, Lukas Hewing, Marco Hutter, and Melanie N Zeilinger. Data-driven model predictive control for trajectory tracking with a robotic arm. *IEEE Robotics and Automation Letters*, 4(4): 3758–3765, 2019.

[55] Joaquin Quinonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *The Journal of Machine Learning Research*, 6:1939–1959, 2005.

[56] Michael Maiworm, Daniel Limon, and Rolf Findeisen. Online learning-based model predictive control with gaussian process models and stability guarantees. *International Journal of Robust and Nonlinear Control*, 31(18):8785–8812, 2021.

[57] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[58] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[59] Marc G Genton. Classes of kernels for machine learning: a statistics perspective. *Journal of machine learning research*, 2(Dec):299–312, 2001.

[60] Joaquin Quinonero-Candela. *Learning with uncertainty: Gaussian processes and relevance vector machines*. PhD thesis, Technical University of Denmark Lyngby, Denmark, 2004.

[61] Andrew James McHutchon et al. *Nonlinear modelling and control using Gaussian processes*. PhD thesis, Citeseer, 2015.

[62] Hassan K Khalil. *Nonlinear systems; 3rd ed.* Prentice-Hall, Upper Saddle River, NJ, 2002. URL `https://cds.cern.ch/record/1173048`. The book can be consulted by contacting: PH-AID: Wallet, Lionel.

[63] Krzysztof Chalupka, Christopher KI Williams, and Iain Murray. A framework for evaluating approximation methods for gaussian process regression. *Journal of Machine Learning Research*, 14:333–350, 2013.

[64] Tilmann Gneiting. Compactly supported correlation functions. *Journal of Multivariate Analysis*, 83(2):493–508, 2002.

[65] Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR, 2009.

[66] Robert B Gramacy and Herbert K H Lee. Bayesian treed gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, 103(483):1119–1130, 2008.

[67] Robert B Gramacy. lagp: large-scale spatial modeling via local approximate gaussian processes in r. *Journal of Statistical Software*, 72:1–46, 2016.

[68] Haitao Liu, Yew-Soon Ong, Xiaobo Shen, and Jianfei Cai. When gaussian process meets big data: A review of scalable gps. *IEEE transactions on neural networks and learning systems*, 31(11):4405–4423, 2020.

[69] Marco F Huber. Recursive gaussian process: On-line regression and learning. *Pattern Recognition Letters*, 45:85–91, 2014.

[70] Hildo Bijl, Jan-Willem van Wingerden, Thomas B Schön, and Michel Verhaegen. Online sparse gaussian process regression using fitc and pitc approximations. *IFAC-PapersOnLine*, 48(28):703–708, 2015.

[71] Thang D Bui, Cuong Nguyen, and Richard E Turner. Streaming sparse gaussian process approximations. *Advances in Neural Information Processing Systems*, 30, 2017.

[72] Samuel Stanton, Wesley Maddox, Ian Delbridge, and Andrew Gordon Wilson. Kernel interpolation for scalable online gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 3133–3141. PMLR, 2021.

[73] Andrew Wilson and Hannes Nickisch. Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *International conference on machine learning*, pages 1775–1784. PMLR, 2015.

[74] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[75] Matteo Saveriano, Yuchao Yin, Pietro Falco, and Dongheui Lee. Data-efficient control policy search using residual dynamics learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4709–4715. IEEE, 2017.

[76] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

[77] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[78] Petros Drineas, Michael W Mahoney, and Nello Cristianini. On the nyström method for approximating a gram matrix for improved kernel-based learning. *journal of machine learning research*, 6(12), 2005.

[79] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.