MINES PARISTECH

MASTERS THESIS

Actively Learning Dynamical Systems with Gaussian Processes

Author:
Mona BUISSON-FENET

Supervisor:
Friedrich SOLOWJOW
Dr. Sebastian TRIMPE

Option MAREVA
Intelligent Control Systems
Max Planck Institute for Intelligent Systems

February 21, 2020

Abstract

Predicting the behavior of complex systems is of great importance in many fields such as engineering, economics or meteorology. The evolution of such systems often follows a certain structure, which can be induced, for example from the laws of physics or of market forces. Mathematically, this structure is often captured by differential equations. The internal functional dependencies, however, are usually unknown. Hence, using machine learning approaches that recreate this structure directly from data is a promising alternative to designing physics-based models. In particular, for high dimensional systems with nonlinear effects, this can be a challenging task.

Learning dynamical systems is different from the classical machine learning tasks, such as image processing, and necessitates different tools. Indeed, dynamical systems can be actuated, often by applying torques or voltages. Hence, the user has a power of decision over the system, and can drive it to certain states by going through the dynamics. Actuating this system generates data, from which a machine learning model of the dynamics can be trained. However, gathering informative data that is representative of the whole state space remains a challenging task. The question of active learning then becomes important: which control inputs should be chosen by the user so that the data generated during an experiment is informative, and enables efficient training of the dynamics model?

In this context, Gaussian processes can be a useful framework for approximating system dynamics. Indeed, they perform well on small and medium sized data sets, as opposed to most other machine learning frameworks. This is particularly important considering data is often costly to generate and process, most of all when producing it involves actuating a complex physical system. Gaussian processes also yield a notion of uncertainty, which indicates how sure the model is about its predictions.

In this work, we investigate in a principled way how to actively learn dynamical systems, by selecting control inputs that generate informative data. We model the system dynamics by a Gaussian process, and use information-theoretic criteria to identify control trajectories that maximize the information gain. Thus, the input space can be explored efficiently, leading to a data-efficient training of the model. We propose several methods, investigate their theoretical properties and compare them extensively in a numerical benchmark. The final method proves to be efficient at generating informative data. Thus, it yields the lowest prediction error with the same amount of samples on most benchmark systems. We propose several variants of this method, allowing the user to trade off computations with prediction accuracy, and show it is versatile enough to take additional objectives into account.

Résumé

De nombreux domaines, allant de l'ingénierie à la météorologie en passant par la finance, nécessitent de prédire le comportement de systèmes complexes. L'évolution de ces systèmes obéit souvent à une certaine structure, qui peut être déduite des lois de la physique ou des marchés par exemple. Cette structure est souvent exprimée mathématiquement par des équations différentielles. Cependant, les dépendences internes des fonctions présentes sont généralement inconnues. L'utilisation d'algorithmes d'apprentissage automatique qui récréent cette structure directement à partir de données est donc une alternative intéressante au design de modèles physiques, qui, encore aujourd'hui, est une tâche difficile.

L'apprentissage de systèmes dynamiques est différent des tâches classiques effectuées par de tels algorithmes, comme l'analyse d'images. En effet, les systèmes dynamiques peuvent être actionnés, souvent par l'application de tensions ou de couples moteur. L'utilisateur a donc un pouvoir de décision sur le système, et peut le commander afin d'atteindre certains états. Actionner le système génère des données, qui peuvent être utilisées pour entraîner un modèle de la dynamique. Cependant, obtenir des données informatives et représentatives de tout l'espace des états est une tâche complexe. Dans ce contexte, la question de l'apprentissage actif prend de l'importance : quelles actions l'utilisateur doit-il choisir pour que les données générées pendant une expérience soient informatives, et permettent d'entraîner efficacement un modèle de la dynamique ?

Pour apprendre des systèmes dynamiques, des outils différents de l'atirail habituel en apprentissage automatique sont nécessaires. Les processus Gaussiens peuvent alors se révéler utiles. En effet, des modèles performants peuvent être entraînés avec des sets de données de taille limitée. Cela est d'autant plus important que ces données sont souvent coûteuses à produire et à traiter, d'autant plus quand il est nécessaire d'actionner un système physique complexe pour les générer. De plus, une notion d'incertitude est comprise dans le modèle, permettant de quantifier à quel point les prédictions sont sûres.

Nous proposons plusieurs méthodes permettant d'apprendre activemement des systèmes dynamiques, par la sélection d'actions informatives. La dynamique du système est modélisée par un processus Gaussien, et l'utilisation de critères provenant de la théorie de l'information permet d'identifier des trajectoires informatives dans l'espace des commandes. Celles-ci permettent d'explorer l'espace des états efficacement, entraînant ainsi le modèle en minimisant la quantité de données nécessaire. En premier lieu, nous analysons les méthodes proposées d'un point de vue théorique, avant de les comparer à l'aide d'un banc de test numérique. La dernière méthode, basée sur du contrôle optimal, se révèle très efficace pour générer des données informatives, et a donc l'erreur de prédiction moyenne la plus basse (à quantité de données fixe) pour la plupart des systèmes que nous testons. Nous proposons plusieurs versions de cette méthode, qui permettent à l'utilisateur d'ajuster l'importance de la la performance du modèle comparé au temps de calcul. Nous montrons aussi que cette méthode est assez versatile pour permettre d'incorporer d'autres objectifs que la simple exploration.

Contacts

Dr. Sebastian TRIMPE trimpe@is.mpg.de
Friedrich SOLOWJOW fsolowjow@is.mpg.de
Mona BUISSON-FENET buissonfenet@is.mpg.de
Intelligent Control Systems Group https://ics.is.mpg.de

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Dynamical systems describe the changing world around us. Controlling their behavior is a difficult, but nonetheless very important problem in many fields such as engineering, physics, and economics. Accurate models are essential for many downstream tasks including control or monitoring, however, it can be difficult to obtain them. First principle models require extensive knowledge about the considered dynamical system. Because of the availability of experimental data, as well as novel models and algorithms to process it, learning dynamics models directly from that data has become an attractive alternative. It is an active research area, with significant interest coming from the robotics community as shown by some recent surveys [1], [2], field in which large amounts of data are often available while some behaviors are very hard to model. The framework of reinforcement learning (RL), historically more focused on learning tasks such as playing games [3], has also been tackling tasks with dynamical systems in the physical world recently [4].

Despite the principled availability of ever more data enabled through modern sensor and computer technology, obtaining rich and informative data sets of dynamical systems in the physical world is still a significant challenge. For example, a humanoid robot that is equipped with hundreds of sensors, each of which generate hundreds to thousands of data points every second, will quickly amass big data. However, if this robot is mostly standing still, this data is essentially useless as it is not informative about the underlying dynamical processes. More generally, generating data sets that are sufficiently rich to train models of a complex dynamical system represents a key challenge in learning for physical system. In addition, generating data on physical processes is often time-consuming and practically involved. This research therefore aims at addressing the question of how to efficiently generate data that is informative for learning of dynamics models. We start by introducing relevant literature on this topic.

Related work When studying dynamical systems such as physical platforms or robots, classical approaches start by building a model of the system dynamics. This model usually depends on certain parameters, which can be unknown. It then becomes necessary to estimate those parameters from data produced by the system, and hence to excite it in a way that the generated data will enable this estimation. This falls into the domain of system identification. For linear time-invariant (LTI) systems, choosing these excitation signals relies on the well-established theory of persistence of excitation [5]. For nonlinear systems however, little results exist and there is no clear method for solving the system identification problem [6]. The related field of Optimal Experiment Design (OED),which aims at easing the task of system identification, is also an active area of research in systems and control [7].

Another way of modeling physical systems is to use data-driven methods, which do not rely on a priori knowledge of the system. Gaussian processes (GPs) are one

of these, and have proven to be an efficient tool for learning systems that are hard to model, such as complex robots [2], friction, contact dynamics, or any other highly nonlinear phenomena. A Gaussian process is a distribution over function spaces: it is a (possibly infinite) collection of random variables, any finite number of which have a joint Gaussian distribution. These models enable efficient regression methods. They are probabilistic, highly flexible, take uncertainty into account, can cope with small datasets, and can incorporate prior knowledge [8], which are useful properties for learning dynamical systems.

Note that hybrid frameworks that combine analytical models and data-driven methods are also an active research area.

A related framework is that of Bayesian optimization (BO). The aim is then not to learn an unknown model underlying some data, but to use this data to estimate the maximum of an unknown function. Gaussian processes have also been used in this context to represent the unknown function [9].

Other works rely on learning how to accomplish a certain task directly by learning a specific policy instead of a dynamics model [4], [10]. These reinforcement learning methods learn to perform actions with dynamical systems, without relying on a model of the dynamics. In this work however, we use model-based methods, which tend to be more general.

In this research, we focus on designing methods that generate informative excitation signals in order to efficiently learn a dynamical system with a GP. Some such methods already exist for learning static GPs, by iteratively choosing where to sample next in order to obtain the most accurate GP model. This is usually done by finding an information-theoretic criterion that guides the sampling procedure, in order to gather data that enables efficient learning. This criterion is often derived from the notion of uncertainty embedded in the GP's posterior variance. For example, [11] uses such an approach to sequentially place a fixed number of sensors in a static field to be learned by a GP, and [12] for studying when samples can be chosen a priori and when they should be adapted during the experiment.

However, active learning for modelling dynamics with GPs is still a widely open problem, though some attempts have been made for example through parametrization of the dynamics [13], [14]. Indeed, in the case of static processes, the active learning strategy mainly needs to compute interesting sampling points to learn from; but in the case of dynamical systems, it is not possible to sample anywhere. Instead, the method will need to decide how to steer the system such that the next states attained will provide interesting samples to learn from.

Contributions In this work, we derive several methods for actively learning a dynamical system with GPs:

- Extension of the state of the art for learning static GPs to the dynamical setting;
- Proposal of a novel input trajectory optimization method, and several computationally more efficient event-triggered approximations;
- Theoretical analysis of each of the new methods, and insights on performance and faced trade-offs;
- Benchmark of the proposed methods on a set of numerical experiments, including robotic systems coming from Reinforcement Learning benchmarks such as OpenAI Gym.

Outline This thesis is organized as follows. We start with preliminary results from the state of the art presented in Chapter 2. We then present notations and a simplified problem formulation in Chapter 3. We design two main active learning strategies for solving the exploration problem in Chapter 4, and analyze them theoretically: besides the classical system identification signals presented in Section 2.3.1, and the greedy method from 3.2, we introduce a separated search and control method inspired from works on active learning for static systems with GPs (see Section 4.1), and an optimal control method (see Section 4.2). In Chapter 5, we compare the proposed methods on a numerical benchmark, where we discuss the experimental set-up and the obtained results, before discussing future work in Chapter 6 and concluding in Chapter 7. The proposed approaches are summarized in Table 4.1.

Chapter 2

Foundations

Before presenting our own concepts and results for actively learning a dynamical system with Gaussian processes, we start with some existing preliminary results. This aims at giving the reader an overview of the state of the art in this field, and tools for understanding the proposed methods. We first introduce some notations, then present the chosen learning framework for this project: Gaussian processes. After stating some of their properties, we present interesting existing results for generating informative data in order to learn a system, some from classical system identification, others from the state of the art on active learning for static GPs. Finally, we also introduce some concepts and works from reinforcement learning (RL), in which similar questions are investigated.

2.1 Context and notation

We consider a system subject to the following discrete-time dynamics:

$$x_{k+1} = f(x_k, u_k) \tag{2.1}$$

$$y_k = x_k + \epsilon_k, \tag{2.2}$$

where f is an unknown Lipschitz-continuous function representing the possibly nonlinear dynamics of the system, $x_k \in \mathcal{X}$ is its state at time $t \in \mathbb{N}$, with $\mathcal{X} \subseteq \mathbb{R}^{d_x}$ the space of possible states, $u_k \in \mathcal{U}$ is the control input, with $\mathcal{U} \subseteq \mathbb{R}^{d_u}$ the space of possible controls, and $\epsilon_k \sim \mathcal{N}(0, \sigma_\epsilon^2)$ is Gaussian measurement noise. For system inputs $X = \begin{pmatrix}(x_0, u_0) & \cdots & (x_n, u_n)\end{pmatrix}^\intercal$, we observe the noisy measurements $Y = \begin{pmatrix}y_1 & \cdots & y_{n+1}\end{pmatrix}^\intercal$. We assume $\mathcal{U}$ is compact, i.e., the possible control input is bounded and closed in each dimension, which is a realistic assumption.

The goal of this thesis is the design of novel strategies for actively learning f: we aim at exciting the system (2.1)-(2.2) such that the data gathered during an experiment yields a sample-efficient estimation of the underlying dynamics. In order to do so, we computethe control inputs $(u_0, ..., u_n)$ that generate informative data for learning f, by optimizing an information criterion. In this chapter, we give an overview of the relevant literature and existing related results. We first present the learning framework chosen for this project: Gaussian processes. Then, we show some existing results on active learning that tackle the question of how to generate informative data that enables efficient learning for the chosen problem. Finally, we also present some aspects of a related field that also deals with learning for dynamical systems: reinforcement learning (RL).

2.2 The learning framework

In this Section, we present the learning method used in this work: Gaussian processes (GPs). We introduce their main features and how to use them, before investigating the literature for actively learning GPs. In this thesis, unless otherwise specified, f represents a true function and $\hat{f}$ its GP estimate.

2.2.1 Gaussian processes for machine learning

The first important piece of literature consists of the book "Gaussian processes for Machine Learning", by Rasmussen and Williams [15]. In Chapter 1, a general introduction is provided and defines broadly Gaussian processes as a generalization of Gaussian distributions to infinite dimensional spaces:

Definition 1. *A Gaussian process is a collection of random variables, any finite number of which follow a joint Gaussian distribution. A GP can be fully characterized by its mean and its symmetric, positive definite covariance function. Assume the process $\hat{f}$ maps the known inputs*

$$X = \begin{pmatrix} x_0 \\ \vdots \\ x_n, \end{pmatrix} \tag{2.3}$$

to the observed outputs

$$Y = \begin{pmatrix} y_0 \\ \vdots \\ y_n \end{pmatrix}. \tag{2.4}$$

Then predictions of $f(x)$ at a new, unobserved point x can be expressed analytically as the posterior mean and variance

$$\mu(x) = k(X,x)^\intercal (K + \sigma_\epsilon^2 I)^{-1} Y \tag{2.5}$$

$$\sigma^2(x) = k(x,x) - k(X,x)^\intercal (K + \sigma_\epsilon^2 I)^{-1} k(X,x), \tag{2.6}$$

where k is the kernel function, $K = (k(x_i, x_j))_{x_i, x_j \in X}$ is the covariance matrix of the data X, and $k(X,x) = (k(x_i,x))_{x_i \in X}$.

In this case, for any fixed possible x, the probability of drawing the associated value $\hat{f}(x)$ follows a normal distribution $\mathcal{N}(\mu(x), \sigma^2(x))$. This regression method using GPs is illustrated in Figure 2.1, where a continuous function is learned by the GP; samples from the prior distribution are learned, then four data points are observed and the posterior distribution is adjusted accordingly.

Note that for a GP with multiple independent outputs, and different kernels for each output dimension, the posterior variance is then $\sigma_i^2(x)$ for each output dimension.

Definition 2. *In the case of d independent output dimensions with d different kernels, instead of the previous posterior variance* (2.6)*, we obtain a posterior covariance matrix of the form*

$$\Sigma(x) = \begin{pmatrix} \sigma_1^2(x) & & 0 \\ & \ddots & \\ 0 & & \sigma_d^2(x) \end{pmatrix}, \tag{2.7}$$

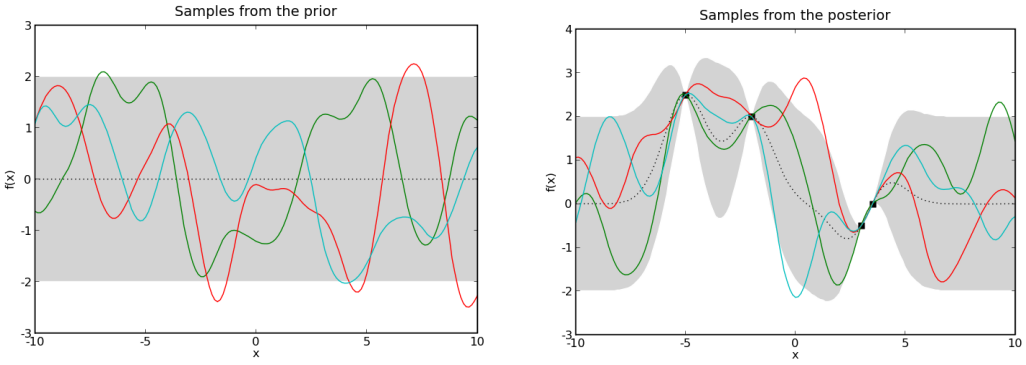


FIGURE 2.1: Learning a continuous function (regression) with a Gaussian process. Samples are drawn from the prior distribution of mean 0, variance 1 (left). Then, four data points are observed by the GP, and samples are drawn from the resulting posterior distribution (right). Source: https://pythonhosted.org.

where $\sigma_i^2(x)$ *is the posterior variance for the* i^{th} *output dimension as per definition* (2.6), *using the* i^{th} *kernel. We then define the scalar posterior variance in the mulit-output case as*

$$\sigma^2(x) = |\Sigma(x)|^{1/d}. \tag{2.8}$$

In this work, we always always use the scalar posterior variance, be it (2.6) for a single output dimension or (2.8) for multiple independent output dimensions.

This type of model is non parametric, which means little assumptions are made on the type of function that best describes the model underlying the data, and therefore the risk of over or under-fitting depending on the class of functions chosen by the user is lower. It is also probabilistic: it provides a notion of the uncertainty of the model. Once the GP has been trained (i.e., data has been observed), its guess for the function f is the mean function μ, and its measure of how accurate this guess is is the posterior variance σ^2.

However, an important design choice remains: the choice of the type of covariance function. The observed data is directly used for predictions through the covariance matrix K: if the new point x is spatially close to an observed point x_i, then they will be considered correlated and the estimation of $f(x)$ will be close to the observation y_i. The main difficulty in training a GP is then to choose the right covariance function, and the right hyperparameters for it: the covariance function will determine the type of relationship between the data points (e.g., the smoothness of $\hat{f}$), and its hyperparameters how closely they are related to each other. GPs are non parametric models, but these design choices still have a great influence on the resulting predictions, and therefore need to be justified and thought through. Some reflections on hyperparameter tuning are therefore given in Chapter 5 of [15]. The relationship between GPs and other commonly used regression tools such as Support Vector Machines (SVMs), Reproducking Kernel Hilbert Spaces (RKHS) and spline models is investigated in Chapter 6 of [15].

Robustness Overfitting is less of a problem than for most machine learning frameworks [16]. Indeed, GPs learning continuous functions with universal kernels (such as the wildly used squared exponential kernel) converge asymptotically: no matter how badly the hyperparameters are chosen, as the amount of data grows to infinity, the GP grows arbitrarily close to the true function [17]. This is not the case for most

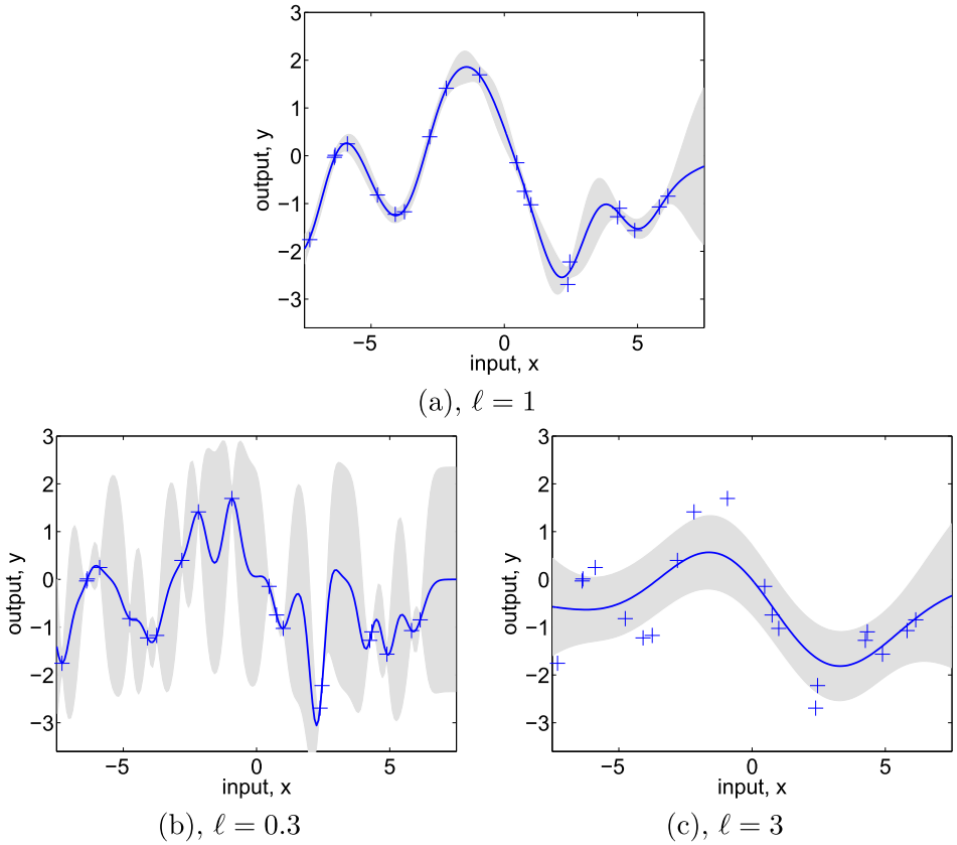


FIGURE 2.2: When performing Gaussian process regression, hyperparameter tuning (choice of covariance function and of its parameters) influences the obtained model. Here, the lengthscale is chosen medium (up), yielding an accurate model, small (bottom left) yielding a model that might not be smooth enough, or large (bottom right), yielding a very smooth model. Source: [15].

parametric models, for which if the class of models chosen by the user to fit the data is wrong, then there is no hope of converging to the true function. The fact that spatial correlations of the prediction point with the measurements are directly taken into account and that there are fewer hyperparameters also helps achieving good convergence in practice. However, for a fixed amount of data, learning performance can still degrade if the kernel or its parameters are badly chosen, for example if the procedure that chooses them by maximizing the log marginal likelihood falls in a bad local optimum. This phenomenon is illustrated in Figure 2.2, where the obtained model for GP regression with small, medium or and large lengthscale in the covariance function are showed. If the lengthscale is large, then the obtained model is much smoother than if it is small; the "best" lengthscale can be determined with more data, or chosen by the user.

Scalability GPs tend to be computationally heavy as adding each new data points requires the inversion of the large covariance matrix K, for which the temporal complexity is $O(n^3)$ with n the number of data points in X. Therefore, GP regression tends not to be scalable and subject to the curse of dimensionality. Making scalable GP models is an active area of research, and several approaches have shown good results. For example, numerous works exist on designing sparse approximations

of GPs that are able to maintain performance while speeding up computations; an overview of the most important approaches is provided in [18]. In future work, it would be interesting to combine such approaches with the methods developed in this thesis, and to evaluate whether the active learning strategies can be sped up enough to run in real-time.

2.2.2 Design choices and limitations

To sum up, we can say that Gaussian processes are a large class of probabilistic, non parametric models. They have been used in numerous works for modeling static processes, and have gained more influence recently for modeling dynamical systems. In this work, we model nonlinear system dynamics with a GP, for which all output dimensions are independent. Since our focus is on exploration and not on the learning procedure itself, we work with the simplest and most common GP training procedures used in literature. One independent GP is learned from all inputs to each output dimension, with stationary hyperparameters that are optimized during the procedure by maximizing the log marginal likelihood. Reasonable prior distributions over the GP's hyperparameters are provided; we call these *hyperpriors*, and guess them from previous experiments. However, this framework might not be ideal for learning nonlinear dynamical systems. For example, some existing approaches learn GP models of time-series dynamics in order to capture delays in the system dynamics, which can yield a significant increase in performance when learning nonlinear dynamical systems [14]. Another issue is having stationary hyperparameters, meaning we assume that the same kernel (with the same lengthscales) can explain the data in all regions of the state space. This assumption is often not verified, and considering nonstaionary GPs, though it leads to a steep increase in the number of hyperparameters, often also increases performance [11] [12] . In the case of nonlinear dynamical systems, whose behavior can vary a lot in different regions of the state space and show delays, considering nonstationary time-series GPs would probably produce much better models. Finally, it is also possible to use coregionalization to correlate the independent output dimensions of the GP models in order to exploit any information the output dimensions could have about each other [19][1]. In this work, for simplicity and in order to focus on the exploration part of the problem, we use the more common framework of an independent, stationary squared exponential kernel for each output dimension.

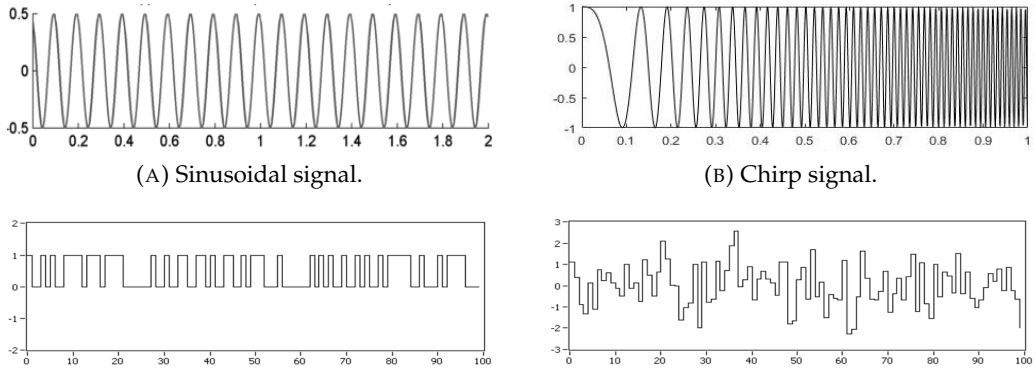
2.3 Generating informative data

After specifying how GPs are used for this research and what their main characteristics are, we tackle the question of how to generate informative data for learning system dynamics with GPs. We give a brief overview of the standard system identification tools often used in this context, then present some previous work on actively learning GPs for static processes.

2.3.1 System identification and notions from classic control theory

Clearly, the problem we tackle in this work, namely informative input generation for dynamical systems, is not entirely new. The field of system identification has been concerned with such questions for some time now. Most methods from this field,

[1] See GPy and a few jupyter notebook tutorials for implementations.



(A) Sinusoidal signal.

(B) Chirp signal.

(C) PBRS

(D) APBRS.

FIGURE 2.3: Excitation signals used in standard system identification. These generic excitation strategies do not usually enable efficient identification of all parameters, most of all for complex nonlinear systems, for which system-specific excitation methods are often necessary. Source: http://www.ni.com/innovations-library/white-papers.

after parametrizing the system dynamics, attempt to identify those parameters by designing "appropriate excitation signals for gathering identification data" [20], which is exactly the topic of active learning. Most of the notions, such as persistence of excitation [5] and experiment design, are well defined and studied for linear systems. However, little results exist on choosing excitation signals for identifying nonlinear systems.

In their survey [6] for example, the authors present a user-oriented guide of the differences between linear and nonlinear system identification, and present their main difficulties. They underline the fact that while solutions of linear systems live on a hyperplane, those of nonlinear systems live on complex manifolds, that are therefore much harder to describe, extrapolate from and navigate; this mathematical difference is at the core of the difficulties experienced in identifying and controlling nonlinear systems. Overall, they insist on the difficulty and the lack of unifying results for nonlinear system identification.

Standard tools exist for supervised input selection for linear systems, but for nonlinear systems it boils down to a complex optimization system, which is the type of approach we will take later in this project. When system-specific signals are avoided, standard signals are usually used, such as: sinusoidal signals of various frequencies, chirps with varying frequency, pseudo binary random sequences (PBRS) or PBRS with varying amplitude (APBRS). See Figure 2.3 for a representation of each of these excitation signals, for which the most important parameters are usually amplitude and frequency (or equivalently minimum time before a change is possible). Later in the project, when benchmarking our methods for actively learning a dynamical system with GPs, we compare them to these standard system identification tools in order to demonstrate their performance.

2.3.2 Active sampling for Gaussian processes

In general, when faced with a machine learning problem, the objective is to reach the desired performance with as little data as possible. Indeed, data is expensive: either because of the cost of conducting experiments, or because of the cost of labeling the necessary amount of data. On top of this, small, highly informative datasets are generally better, because they can contain the same amount of information as a large

dataset while being less costly to obtain and easier to do computations with. Such ways of thought lead to the concept of active learning, where the aim is to decide during the learning procedure which point should be sampled next, depending on what has been seen until now. This is also used in other domains, such as image classification: during training, the performance of the classifier for each possible class is monitored, and the next set of training images is chosen primarily with samples of classes that have not been learned well yet, in order to enhance efficiency of the training procedure.

Most machine learning paradigms are faced with this issue, and designing methods that decide where to sample next in order to obtain an informative dataset is an active area of research. Some approaches already exist for learning static GPs. We present them in the next paragraph, before introducing some further work on exploration with dynamic GPs. But first, we introduce some of the information criteria often used for actively learning GPs.

Types of information criteria Before presenting some existing methods for actively learning static GPs, we define the information-theoretic criteria often used in this context.

- Posterior variance:
  In the Bayesian framework, which is the case for GPs, we have a prior distribution before observing the data, and then a posterior distribution after having observed it. We have the set of sample points X and the set of observations $Y = f(X) + \epsilon$ with ϵ a Gaussian noise vector of variance σ_ϵ^2, where f is a GP. For a new test point x (or set of test points), we assume a certain prior distribution of $\hat{f}(x)$, usually of the form $f(x) \sim \mathcal{N}(0, \Sigma_{prior})$. Then, using the GP trained on the observations, we obtain a posterior distribution of mean (2.5) and variance (2.6), as in Definition 1.

  In this case, σ_{prior} is the prior variance and σ^2 is the posterior variance. Therefore, maximizing the posterior variance boils down to choosing x the new test point such that the uncertainty at that point, named $\sigma^2(x)$, is highest.

- Entropy:
  The entropy of a random variable quantifies the uncertainty we have on that random variable. In the context of GPs, we define the notion of differential entropy.

  Definition 3. *The differential entropy of a new sample point x_*, given a GP characterized by its posterior mean and variance functions $(\mu(\cdot), \sigma^2(\cdot))$, is given by*

  $$H(x_*) = \frac{1}{2}\log(2\pi e\sigma^2(x_*)). \tag{2.9}$$

  With this definition, maximizing the posterior variance and maximizing the entropy are equivalent, however, the entropy is numerically more tractable thanks to the log term, that making it easy to compute even for small values of the variance.

- Mutual information:
  Mutual information measures how much information two random variables share, i.e., how much knowing about one of the variables reveals about the other.

Definition 4. *The mutual information between two random variables X_1 and X_2 is defined as*

$$MI(X_1, X_2) = H(X_1) - H(X_1|X_2) = H(X_2) - H(X_2|X_1), \tag{2.10}$$

where $X_1|X_2$ denotes that X_1 is conditioned on X_2.

In the case of optimal sensor placement [21], it is defined as

$$MI(S \setminus X, X) = H(S \setminus X) - H(S \setminus X|X) \tag{2.11}$$

with X the sampling points and S the space on which $\hat{f}$ is defined. Maximizing this criterion finds the sampling set X which makes the entropy of the unsampled space as small as possible compared to before, i.e., that maximizes the information about $S \setminus X$ given by X.

Note however that this same mutual information criterion can be given different meanings depending on the variables that are considered in it. As shown in [9], using our own notations, when looking at the mutual information criterion $I(Y, f) = H(Y) - H(Y|f)$, we have

$$H(Y) = \frac{1}{2}\log|2\pi e(K + \sigma_\epsilon^2 I)|; \qquad H(Y|f) = \frac{1}{2}\log|2\pi e\sigma_\epsilon^2 I|, \tag{2.12}$$

which yields

$$MI(Y, f) = \frac{1}{2}\log\left|I + \frac{1}{\sigma_\epsilon^2}K\right|. \tag{2.13}$$

In this case, maximizing the mutual information corresponds to maximizing the knowledge about f that is given by the observations Y_X, which here is equivalent to maximizing posterior variance. In this work, we focus on the differential entropy criterion H, since this is equivalent to posterior variance but numerically more tractable.

In this work, we focus on the differential entropy criterion H, since all three are equivalent in our settings, but entropy is more tractable numerically.

Active sampling for static Gaussian processes There are some examples of active learning frameworks for Gaussian processes in literature. Most of them are concerned with learning a static phenomenon with a GP. One of the most well-known examples, and very useful for our future work, is the sensor placement problem [11].

Imagine that a GP is used to estimate the temperature f along a line: $f(x)$ denotes the temperature at distance $x \in \mathcal{X}$ from the origin of the line. The GP $\hat{f}$ is used to approximate f; some data has previously been collected, and now N sensors are available for taking measurements. The aim is now to find the most informative N sampling locations at which to place the sensors, i.e., which will yield the best estimation of f for a fixed number of sampling points. This can be done by finding the locations that maximize a certain information criterion I, which can for example be differential entropy (see Definition 3) or mutual information (see Definition 4) for example (in this case, the authors argue that mutual information yields better performance). Since maximizing such a criterion is NP-hard, the authors propose the

following greedy rule:

$$x_i = \underset{x \in \mathcal{X} \setminus \{x_1,...,x_{i-1}\}}{\operatorname{argmax}} I(x) \; \forall \, i \in \{1, ..., N\}, \tag{2.14}$$

At each iteration, the information criterion is maximized and x_i is chosen as the next sampling point. Thanks to a property of the information criterion I called *submodularity*, which we will consider more in detail later in this thesis (see 4.1.1), it can be shown that the sequence $(x_1, ..., x_N)$ of sampling points selected by (2.14) is worse than the optimal sequence only up to a constant bound. Furthermore, if the information criterion I is one of the aforementioned ones (see 2.3.2), then it only depends on the posterior variance of the GP and not on its mean; hence, with (2.5), it does not depend on the observed data Y. This means that it is possible, for fixed kernel and covariance matrix given by previous observations or expert knowledge, to compute the $(i+1)^{\text{th}}$ greedy placement without having actually taken any measurements with the i first sensors, just by knowing where they are placed. Hence, $(x_1, ..., x_N)$ can be computed offline: it is possible to first compute where to place the N sensors, then place and activate them all at once, without needing to first place each, take a measurement with it, then decide on the next one.

Several papers build on these results for the sensor placement problem. For example in [12], where the authors investigate when sequential placement is advantageous compared to a priori placement. Indeed, as we just stated, the greedy rule (2.14), $(x_1, ..., x_N)$ can be computed offline, i.e., a priori, without seeing any data from the sensors (except some already available data to start with). But this is only true if the GP hyperparameters stay the same during the procedure. Indeed, if the user is not sure of the hyperparameters and wants to optimize them after the i^{th} sensor was placed, in order to have a more accurate estimation of f and therefore a better placement for the $(i+1)^{\text{th}}$ sensor, then it becomes necessary to actually make a measurement at x_i, optimize the hyperparameters, and compute x_{i+1} accordingly. This is called the *sequential* approach, as opposed to the previous *a priori* approach. In [12], the increase of performance given by the sequential approach is compared to its cost. A rule for switching from sequential to a priori is derived, depending on how high the uncertainty over the hyperparameters is: once they are well-known and the model fits well, there is no need for optimizing them and the a priori method can be used. This trade-off between a priori and sequential sampling seems very close to the well-known exploration/exploitation trade-off in reinforcement learning: there is always some interest in staying in a known regions, but also in traveling to regions where the estimate is highly uncertain, since they have a high information gain potential. These two sides often have to be weighed against each other.

A related problem is that of Bayesian optimization (BO). In this case, the aim is not to estimate f as accurately as possible, but to estimate its maximum (or respectively minimum). Using as few noisy observations as possible, we want to find

$$x_* = \underset{x \in \mathcal{X}}{\operatorname{argmax}} f(x). \tag{2.15}$$

This can be illustrated with the same use case as for the greedy active learning strategy: image we have N temperature sensors, some previously collected data or expert knowledge, and we want to find the maximum temperature along a line. We again have to decide where to place the N sensors, but the aim is different: instead of trying to get an overall good model $\hat{f}$, we want to estimate x_*. The authors of [22] propose the GP-UCB strategy: at each iteration, the next sampling point x_i is chosen

as

$$x_i = \underset{x \in \mathcal{X}}{\operatorname{argmax}}\, \mu_{i-1}(x) + \beta_i^{1/2} \sigma_{i-1}^2(x), \tag{2.16}$$

where μ_i is the posterior mean of the GP and σ_i^2 its posterior variance at iteration i. In this case, both the posterior mean and variance are taken into account, and the parameter β is used for weighing off whether the focus is on exploitation (choose x_i close to places where the posterior mean is high) or on exploration (choose x_i close to places where the uncertainty is high). Since the posterior mean intervenes in (2.16), and the observations Y are necessary to compute it (see (2.5)), this method can only be applied sequentially and not a priori (a measurement from each sensor is necessary to compute the next placement).

Other works based on similar ideas include [13], where the near-optimal sensor problem [11] is extended to near-optimal path planning for an underwater robot. The space to explore is discretized into a finite number of possible waypoints to visit, linked by a finite number of straight paths from one waypoint to the other. The information criterion in (2.14) is then parametrized by the path P, an ordered list of waypoints to visit. The resulting problem is very close to the sensor placement formulation, since it consists in optimizing over a list of possible locations on a grid, and can be solved efficiently by using results from combinatorial optimization. Similar ideas are used for informative path planning of multiple underwater robots in [23], but no notion of dynamics is introduced there either.

The previous active learning methods rely on the fact that f is static: if a sampling location x is chosen, then a sensor can simply be placed there and a new data point is produced right away. This is however not the case for dynamical systems: in a dynamical system, if we wish to observe the state x, then we have to drive the system to that state by controlling it. The dynamics can be understood as constraints for the active learning problem. Though some papers relying on similar ideas for control systems exist ([24], [25]), there are to the best of our knowledge no unifying results extending such approaches to dynamic GPs. In the next chapters, we will derive several novel methods for actively learning dynamic GPs, some of which are extensions of the previous results to the dynamical setting. We will show this extension is a first step in solving this problem, but sub-optimal. We will then design an active learning strategy that takes the dynamics directly into account and shows superior performance on a numerical benchmark. But first, let us introduce some related work on using GPs to learn dynamical systems.

Exploration and control with dynamic Gaussian process models As presented previously, most existing work on the topic of active learning for GP models focuses on static phenomena. There are however some results on exploration and control of dynamical systems with GPs. In [8] for example, the authors model a dynamical system with a GP, not by using input-output data directly, but rather by learning a time-series model of the dynamics. They provide a greedy scheme for generating informative control inputs that reduce the uncertainty about the GP hyperparameters, and a control scheme inspired from model-predictive control (MPC) for driving the system to follow a reference trajectory once the GP has be learned reasonably well. They also investigate how to choose the most informative subset of already sampled data to keep in memory while the system keeps running, and only gets re-learned and updated once in a while; this is also interesting, but further away from our direct concerns. Their informative control generating strategy resembles the greedy

scheme we propose in the next chapter (Section 3.2), and their MPC-inspired control scheme for the GP yields an optimization problem that is similar to the active learning formulation (see (3.5)-(3.6)), but with a different objective.

Exploring with a dynamical system while respecting certain safety constraints is also an active area of research, and some of the existing results use GP models. This makes sense for industrial applications, since systems often have a certain unknown safe subspace in which they can evolve, but can get damaged when they get out of the safe zone. In [26] for example, the settings are quite similar to the active learning problem for dynamic GPs: the aim is to learn f the true dynamics of a system with a GP, by using a control policy π to excite it. In order to learn f, it is necessary to explore the state-space, but the authors incorporate safety constraints by restricting the space that can be explored to safe ellipses. The designed controller is an MPC that optimizes a cost function containing an information criterion, but subject to the constraints of staying in the safe space. They estimate the state in T time steps through a propagation of the current GP model through time, and a trajectory is only considered acceptable (i.e., respecting the safety constraints) if the ellipsoid representing the confidence region of the state in T time steps is still in the safe space. Though their paper focuses a lot on safety, which is out of scope for this thesis, the adopted approach is similar to the active learning problem we will formulate in the next chapter (see (3.5)-(3.6)), with some differences in the assumptions that are made (GP is learned on residuals only, assuming affine state-feedback control).

The problem formulation in [14] is also similar to the active learning problem, with a focus on safety. Two GPs are trained: one that models the dynamics of a system that we want to learn, and one that models the safety of a given state. Both are learned on data collected from the physical system, i.e., state and control input data measured on the system, but also from a safety information that the system provides, for example the temperature of a critical component. The GP is again not trained directly with input-output data but with time series following a non exogenous model. The authors parametrize the system trajectories as piece-wise constant, then optimize this parametrization to maximize an information criterion while respecting a certain probabilistic safety constraint. Therefore, their framework does not directly choose the most informative control inputs to send to the system, but the whole most informative parametrized trajectory in input space. Hence, in this work also the considered problem formulation is similar to (3.5)-(3.6), but the assumptions and proposed methods differ.

All in all, though some recent works have started going in that direction, we still know of no general methods for actively learning dynamic GPs. We will propose such methods in the next chapters, after introducing some related work on reinforcement learning.

2.4 A related paradigm: reinforcement learning

In the field of reinforcement learning (further RL), an agent learns to perform a certain task by interacting with its environment. Some of the recent research attempts to apply such approaches to robots and other types of dynamical systems. Exploration of the state space is therefore also considered in RL, and weighed off compared to exploitation of the knowledge already gathered, leading to the well-known trade-off between exploration and exploitation. Though active learning is not explicitly a research topic in RL, since the focus is more on the task that an agent wants to accomplish, the problems we face and concepts we propose are related. Some of the

recent RL frameworks and papers for learning to perform a task in a physical system are presented in the next section.

2.4.1 Basic concepts for reinforcement learning

Reinforcement learning is based on an agent learning how to perform certain tasks by trial-and-error, by interacting with its environment. We have S a state space representing the agent's environment, A a set of possible actions, $R(s,a)$ a reward function associated with state s and action a, and $T(s'|s,a)$ the transition probability of getting to state s' when performing the action a in state$s$. The environment is considered non deterministic since performing action a in state s does not need to always lead to the same outcome s'. The most common framework for RL is that of Markov Decisions Processes (further MDP): we consider that the probability of getting to state s' after performing action a only depends on the current state s and not of the previous states and actions, i.e., the system has no memory of previous states and actions.

The reward and transition functions, respectively R and T, are considered unknown to the agent, who can interact with its environment in order to learn them. In other words, we do not know how our actions impact the world, and we do not know the immediate reward after an action; if we did, then finding the sequence of actions that best performs a certain task would be a planning problem and would not necessitate any interaction, but only the optimization of a certain criterion under this sequence of possible actions. The objective of the agent is then to learn a policy, i.e., a mapping that associates to each possible state an action to be taken by the agent, that enables it to perform the task. The agent is guided in the training process by the reward function, which returns high rewards if the agent is getting closer to achieving the task, and low rewards if it is getting further. There are two main ways of obtaining such policy: either policy search (finding π^* the policy which maximizes the sum of future expected rewards) or value iteration (finding the optimal value function V^* which maximizes the sum of future expected values, and from which we can deduce an optimal policy).

Two types of RL should be distinguished. On the one hand model-based RL, where the agent first tries to learn a model of its environment (learn the functions R and T), then plans its policy using those estimates. On the other hand model-free RL, where the agent directly evaluates a policy or a value function and tries to optimize them. According to literature [4], [27], model-based RL has proven to be more efficient in learning control of dynamic systems for now, since it can leverage the agent's knowledge of its environment in order to find the optimal policy, and therefore needs fewer iterations (i.e., fewer expensive interactions with the environment) before it can perform the given task.

2.4.2 Policy search for Gaussian processes

In [4], the authors introduce PILCO, a Matlab framework that aims at learning an optimal policy for a dynamical system modeled by a GP, using reinforcement learning. This framework performs policy search in a data-efficient way: it uses gradient descent to optimize the parameters θ of the control policy $\pi(x_{k-1},\theta) = u_{k-1}$ according to a cost function $J^\pi(\theta)$ by using responses of the system to inputs as data points, while keeping the number of interactions as low as possible. This gradient descent can be computed analytically using the expressions of posterior mean and variance for a GP, along with moment matching for uncertainty propagation. Note

that uncertainty propagation in this context means propagating the GP in time. Take a simple example: assume (x_k) is a sequence of random variables such that $x_{k+1} = f(x_k)$ is normally distributed for a fixed k. If f is a nonlinear function, then the random variable x_{k+2} is not normally distributed anymore. But it can be approximated by a Gaussian distribution using approximation techniques, such as moment matching. There exists a rich literature on the design and evaluation of such approximation methods, that enable the propagation of a GP through nonlinear transformations.

PILCO iteratively performs the following steps until a task is considered successfully learned:

- Train a GP model of the environment from all recorded data
- Approximate the cost function $J^\pi(\theta)$
- Perform gradient descent until a minimum of $J^\pi(\theta)$ is reached at θ^*
- Set the control policy to $\pi^* = \pi(\theta^*)$
- Evaluate π^* by applying it to the system: the agent receives a reward on π^* and new sample points
- Iterate until the reward on π^* is considered sufficient.

This method is particularly useful for robotics, since it can derive a good policy π in only a few number of iterations, i.e., a few number of experiments, which tend to be expensive for robotic applications in particular.

Several extensions and improvements of PILCO have been proposed in literature. In [27], it is made clear that he classic assumptions for GP modeling are that the inputs X are noise-free, and the outputs $Y = f(X) + \epsilon$ have i.i.d. noise, but that both these assumptions are violated when modeling time series such as the state of a dynamic system. The authors then propose a method to leverage these difficulties: training the GP to predict long-term trajectories instead of producing only one-step ahead predictions, so that the success of the task that is being learned is taken into account directly into the model training. This approach can be implemented directly into PILCO by replacing the usual hyperparameter optimization with a more complex mechanism, that takes the long-term prediction error into account. In the same line of thought, [28] also proposes hyperparameter optimization and uncertainty propagation procedures that are more adapted to learning system dynamics with a GP, most of all when latent states and input and output noise are present. As presented in these papers, there are more sophisticated ways of learning a dynamic GP and propagating it through time than what we use in this project. Since our focus lies on exploration, we use the more standard frameworks for learning $\hat{f}$, and leave combining the methods presented in this thesis with more advanced GP training schemes for future work.

Note also that PILCO, like any RL framework, is task-specific: a policy is only learned for a specific task, and has to be re-learned if the task changes even slightly. Most scientists in favor of model-based RL argue that learning a model, then using it to derive a policy is more general since the model can then be re-used for another task. This also holds true for active learning in general: learning a model is a high-level objective, and if a good model can be learned from data, then it can be used for control and accomplish various tasks. This is why, in this work, we focus on actively learning a GP model instead of a specific task.

2.4.3 Explicit exploration for reinforcement learning

The concept of exploration has also been investigated in the context of RL. In [29] for example, the authors accelerate Q-learning on several classic benchmarking tasks (including the Gym tasks from OpenAI) by designing methods for global exploration of the policy space. The criticality of exploration in the case of DDPG, the adaptation of Q-learning to continuous state space, is underlined; it is mentioned that most exploration strategies used until now are rather simplistic (simply adding uncorrelated noise to the policy outputs). Since DDPG is an off-policy learning scheme, it is possible to design a teacher policy that explores the state space and gathers informative data, and a student policy that uses the collected data to optimize the true RL policy of the agent. This teacher policy will then explore regions of the state space that have high improvement expectation (expected to increase the value of the student policy the most based on past experiments), and pass on this exploration data to the student policy, which simply follows deterministic DDPG based on the data gather until now. The authors show significant improvement of the quality of the policy compared to classic DDPG, but also significant computational overhead.

In [30], the focus lies on designing robust control policies that minimize a quadratic cost for linear systems with unknown parameters and static-gain feedback control. The authors design control policies that jointly minimize the worst-case cost given current uncertainty about the system (worst-case for robustness), and the uncertainty as to whether this cost is indeed worst-case. The empirical results show that the obtained control policy is more efficient than a nominal controller, because it has learned more about the environment, and also more than a greedy controller, because even though the greedy exploration method often learned more about the system, it also learned parameters that may be useless for the control task. The same authors develop this approach again for a more control-oriented audience in [31].

These recent works on exploration for reinforcement learning underline how critical it can be for learning frameworks to explore well, be it exploring the input space for learning a dynamics model, or exploring the policy space for learning a task-specific policy. This shows that the topics of exploration, of data-efficiency and hence of active learning spark more and more interest in the machine learning research community. The active learning methods proposed in this thesis are part of this broader research topic.

Chapter 3

Problem formulation and greedy method

In this chapter, we present the first ideas and numerical experiments for actively learning dynamical systems with GPs. We start with some notations, then introduce our main ideas for the dynamical active learning strategy, and derive a simplified problem formulation, with some numerical experiments as a proof of concept. In the next chapters, we will extend this to a complete problem formulation and present different active learning strategies, which we will then benchmark.

3.1 General formulation

We consider a system subject to the following discrete-time dynamics, starting from fixed initial state $x_0 \in \mathcal{X}$:

$$x_{k+1} = f(x_k, u_k) \tag{3.1}$$
$$y_k = x_k + \epsilon_t, \tag{3.2}$$

as defined in 2.1. The true function f is modeled by a GP $\hat{f}$. This GP maps inputs $\begin{pmatrix}(x_0, u_0) & ... & (x_n, u_n)\end{pmatrix}^\intercal$ to outputs $\begin{pmatrix}x_1 & ... & x_{n+1}\end{pmatrix}^\intercal$. We denote X the dataset available for training, and Y the set of corresponding noisy measurements of the states $(x_1, ..., x_{n+1})$. Its posterior mean and variance can be expressed as

$$\mu(x) = k(X, x)^\intercal (K + \sigma_\epsilon^2 I)^{-1} Y \tag{3.3}$$
$$\sigma^2(x) = k(x, x) - k(X, x)^\intercal (K + \sigma_\epsilon^2 I)^{-1} k(X, x), \tag{3.4}$$

where $K = (k(x_i, x_j))_{x_i x_j \in X}$ is the covariance matrix of the dataset X, and $k(X, x) = (k(x_i, x))_{x_i \in X}$. We denote $\hat{f}_k$ the GP after having observed $y_k = x_k + \epsilon_k$, H_k its entropy function, and so on.

The GP can be evaluated at any new test point (x_k, u_k), i.e., predict the next point x_{k+1}, through the posterior mean $\mu(x_k, u_k)$ and the posterior variance $\sigma^2(x_k, u_k)$ at this point. We assume the system is reasonably stable and controllable in the exploration region, in the sense that it does not diverge, and it can be actuated. This is the case for example with robots. Indeed, if exploration can cause the system to diverge, or if the control inputs do not influence it, then the problem is ill-posed, and there is little any active learning strategy can do.

In this work, our focus is on exploration and not on the learning procedure itself. Therefore, we work with the simplest and most common GP training procedures used in literature, as described in Section 2.2.2. In the experiments, we mostly use the standard squared exponential (SE) kernel.

The active learning problem can be stated as follows: we wish to learn $\hat{f}$ as efficiently as possible, i.e., for a fixed number of sample points, to be able to approximate the unknown system dynamics f with $\hat{f}$ as closely as possible. Hence, we want to explore the input space so that the dataset used for training the GP is highly informative. This task can be formulated as an optimization problem:

$$\min_{x\in\mathcal{X},u\in\mathcal{U}} J((x_k,u_k),...,(\hat{x}_{k+M},u_{k+M})) \tag{3.5}$$

$$s.t. \quad \hat{x}_{k+1} = \hat{f}(x_k,u_k)\ \forall\ k \in \{k...,k+M\}, \tag{3.6}$$

where J is a cost function, and M the time horizon, which is assumed to be fixed. In this formulation, there are two optimization variables: the state x, and the control input u, which need to be optimized jointly since they are not independent. However, for a well-posed dynamical system, given the dynamics, an initial position x_k and a controller u_k, there is a uniquely defined "arrival" point x_{k+1}; so we can optimize a function of x_{k+1} by just acting on the optimizing variable u_k. Hence, we can re-write (3.5)-(3.6) as:

$$U_k^* = \underset{u_k,...,u_{k+M-1}\in\mathcal{U}^M}{\operatorname{argmin}} J((x_k,u_k),(\hat{f}(x_k,u_k),u_{k+1}),(\hat{f}(\hat{f}(x_k,u_k),u_{k+1}),u_{k+2}),...), \tag{3.7}$$

where $U_k^* = (u_k^*,...,u_{k+M-1}^*)$, and the only optimization variable is u, and not x. After each iteration, a control trajectory is applied and yields new observations. We use these observations to update the GP and optimize its hyperparameters. After that, we restart a new optimization procedure. Indeed, if we did not update the GP after seeing data, then we might as well do the optimization procedure a priori, i.e., blindly, since the cost does not usually depend on the observations directly. This is the general scheme we investigate for generating informative control inputs that enable data-efficient training of the GP; we will provide more specific results bit by bit over the next chapters.

3.2 Simplified problem: the greedy method

The first method we propose is obtained by simplifying the problem somewhat: we consider $M = 1$ (one-step ahead predictions), and for the cost at time k with a fixed x_k, we take $J = -H_k(x_k,u_k)$. With this simplified formulation, from fixed x_k, we select the next control input according to the greedy rule

$$u_k^* = \max_{u_k\in\mathcal{U}} H_k(x_k,u_k). \tag{3.8}$$

This approach aims at selecting control inputs that yield $\hat{x}_{k+1} = \hat{f}(x_k,u_k)$ with high uncertainty, i.e., control inputs for which the GP cannot predict the output accurately, given the current state. Hence, the selected u_k explore both the state space (select u_k such that the next state x_{k+1} is in an unknown region of $\mathcal{X}$) and the input space (select u_k that have not been used before). This enables the model-based exploration of both $\mathcal{X}$ and $\mathcal{U}$, in order to generate an informative dataset for training $\hat{f}$.

3.2.1 Implementation and tests

We start by implementing the greedy method and investigating the obtained results. At each time step k, given the current state x_k, we choose u_k according to (3.8), using

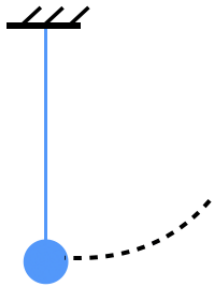
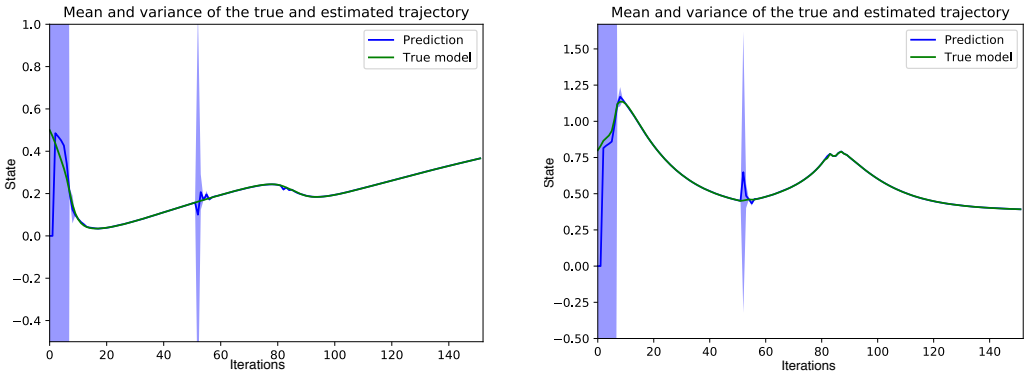


FIGURE 3.1: Schematics of an inverted pendulum.

`scipy`'s library `optimize.minimize`. Then, we estimate the next state $\hat{x}_{k+1}$ given the selected u_k, measure the true next state x_{k+1}, and plot both. We implement the dynamics of a simple inverted pendulum, discretized and solved with a Runge-Kutta 4 (further RK4) scheme [32], and investigate several aspects of our greedy solution on this simple example.

We use `scipy`'s `optimize.minimize` library for this first method, which might not be ideal in the future for running experiments on active learning with GPs for dynamical systems. Indeed, the optimizers provided there, such as BFGS or Nelder-Mead methods, assume convex and sufficiently smooth objective functions. This is a classic choice in optimization, but not the case for objective functions that involve posterior variance or entropy of a GP. Therefore, the provided optimizers will only deliver local optima and might get stuck in them, for example if the initial point for the optimization does not change. One possibility is then to try out random initializations and pick the lowest cost to start the optimization procedure. We also test initializing the optimization procedure either at 0, or at u_{k-1}. The "random best" initialization produces the best results, since it helps finding a better local optimum, and it is the one we use for further experiments. For future experiments, we will use more advanced optimization toolboxes. These plots mostly serve as a proof of concept for the general ideas and as base-case results for the greedy method.

Dynamics tested We simulate a simple inverted pendulum as the first system and test our methods with it. This is a good system to start with, since it is easy to understand and interpret; a schematics is shown in Figure 3.1. There are two states, the pendulum angle θ and its angular velocity $\dot{\theta}$. The dynamics have an auto-regressive structure, and the ODE on $\dot{\theta}$ is nonlinear because of a $\sin(\theta)$ term, which can be considered as linear locally around the stable equilibrium $\theta = 0$, when the pendulum is handing down. Precise dynamics equations and parameters of all studied systems are listed in Section 5.1.1. We plot the accumulated angle of the pendulum ($\theta = 3.14$ represents one swing-up around its axis, after which the angle keeps growing), and use a squared exponential kernel for learning. Note that, even if we can see on the graphs shown here that one step ahead predictions are accurate after only 15 time steps, this does not mean that the learned model has high quality. Indeed, if the system stays in a small region of the state space: as seen in Figure 3.4, we have no information on how accurate the model is when looking at a larger portion of the state space. We are also only plotting one step ahead predictions and not several steps ahead. Metrics for measuring how much of the state space was explored with a certain active learning strategy and how well the corresponding system could be learned are given in Section 5.1.3, in order to quantitatively compare the different methods.



(A) Trajectory of x_1 (B) Trajectory of x_2

FIGURE 3.2: Actively learning the dynamics of a continuously-stirred reservoir tank [33] (CSTR) in simulation using the greedy method (3.8) with bounded control. We plot the estimated states over time (blue line), along with the posterior variance (light blue), against the true states (green line). After about 15 iterations, the predictions are very close to the true next state, and the variance is very low: the GP has learned the dynamics of the system in this region of the state space.

We also implement simulations of a nonlinear chemical system described in [33], Section 9. This time, the dynamics are exponential and more complex than for the pendulum. They are also presented in details in Section 5.1.1, and illustrated in Figure 3.2.

Bounding control and state Actively learning a dynamic GP boils down to exploring its input space. If the control input is left unbounded, then it tends to grow to infinity in order to always explore further, as seen in Figure 3.3, where the accumulated angle of the pendulum simulations is diverging (pendulum keeps turning around). Assuming limited control is a reasonable assumption, since this is often the case in practical applications, and it makes exploration more challenging. For all further experiments with the pendulum, we use $\mathcal{U} = [-5, 5]$. With these bounded inputs, the system stops diverging and we obtain a more useful oscillatory behavior, as shown in Figure 3.4. In some applications, it can also make sense to bound $\mathcal{X}$ to enforce hard constraints on the states.

3.2.2 Influence of the cost function

For now, we have only considered a purely information-theoretical criterion, i.e., minimizing: $J_t(u_k) = -H_k(x_k, u_k)$. However, it could be useful to add other terms to the cost function for certain applications: a more complex cost function can take other objectives into account as only active learning, such as achieving a specific task, minimizing control and sensing costs, etc. This is the case for example in Bayesian optimization (see Section 2.3.2), where the objective is not to learn an unknown function f, but to find its maximum. Adding a term of type $-\alpha\mu(x_k, u_k)$ to (3.8) could incorporate this extra objective, similarly to the GP-UCB method for Bayesian optimization in the static case [22]. We leave such possible extensions for future work.

We start by adding a term in the cost function, weighted by $\alpha > 0$:

$$J_k(x_k, u_k) = -[H_k(x_k, u_k) + \alpha||u_k||_{L^2}]. \tag{3.9}$$

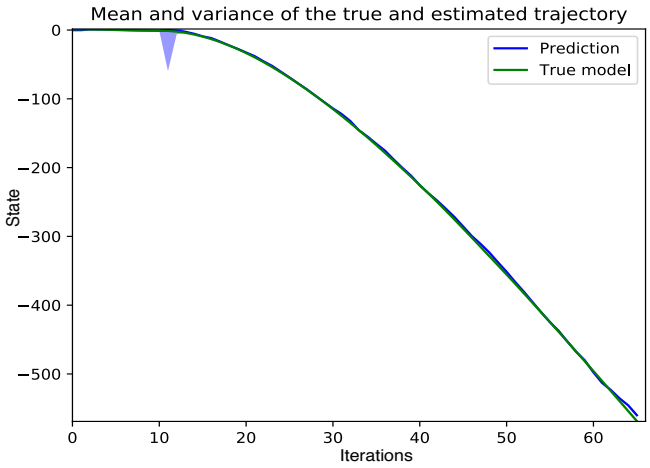


FIGURE 3.3: Actively learning the dynamics of a pendulum in simulation with the greedy method (3.8), using the true dynamics for active sampling. We plot the estimated states over time (blue line), along with the posterior variance (light blue), against the true state (green line). With no constraints on the value of u, we observe that the active sampling strategy drives the control to high values, and the system naturally diverges.

Adding this term reduces the control costs of the active learning procedure. It also makes sure that, even if states often bear more uncertainty than controls, which can push the system towards always using the highest possible control values to reach new states, intermediate control values are also generated, which can sometimes increase learning performance. Finally, it also serves as a type of regularization, similarly to L^2- regularization often used in machine learning: it makes the control trajectory somewhat smoother, and therefore the mapping from state and control to next state somewhat easier to learn. It also comes with a weighing parameter α, that easily enables the user to decide how much smoothing should be done. We typically used values of α around 10^{-2} for our experiments. Though we end up focusing on the entropy criterion only, since it is the most direct one for active learning, this serves as a proof that adding other terms to the cost considered in our optimal control methods is very possible, and can increase performance in some use cases.

In the pendulum simulation with bounded control, this cost function yields different behavior, but also enables learning f. As illustrated by Figure 3.5, one exploration trajectory (top) has been obtained with entropy, the other (bottom) by adding the norm of u_k in the cost. In the top figure, we obtain highly varying control inputs, whereas in the bottom one the control is very high at the beginning but quickly stabilizes around 0, yielding a lower control cost in terms of its norm over time. This behavior exhibits a trade-off between information gain and control costs. It also works as a natural stopping criterion: once the uncertainty is below a certain threshold, then gaining more information is not worth it compared to the costs it brings, therefore the control input converges to zero; we could use that in the future as a stopping criterion. However, we should not overuse it, since it greatly constrains the exploration possibilities of the strategy and yields less variety and less amplitude in the possible behaviors. In the final experiments, we end up using only the information criterion without any extra cost, since this slows down the exploration procedure; but it can still be useful for some applications.

Similar behavior is observed when testing the control cost (3.9) with the CSTR simulations, as seen in Figure 3.6 (here on x_2). Here, we test different values of the

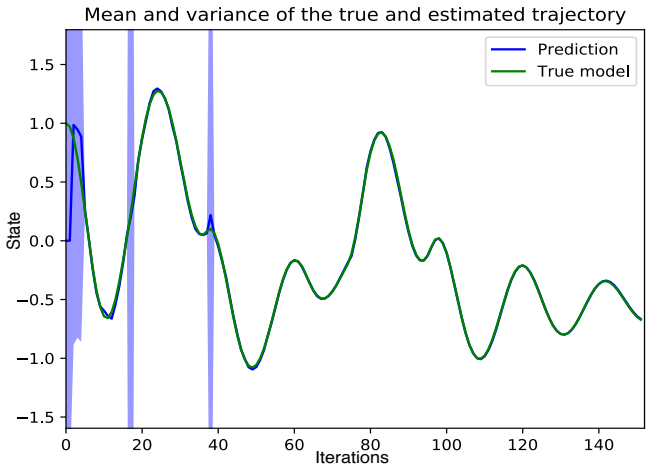


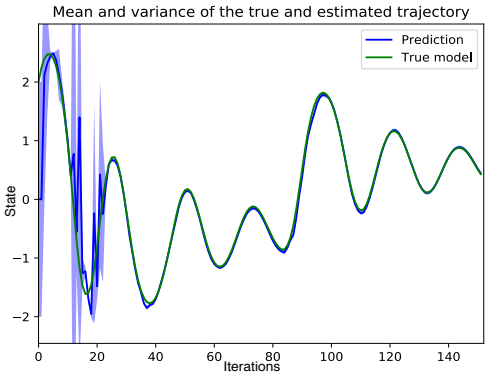
FIGURE 3.4: Actively learning the dynamics of a pendulum in simulation using the greedy method with bounded control. We plot the estimated accumulated angle over time (blue line), along with the posterior variance (light blue), against the true angle (green line). After about 15 iterations, the GP has learned the dynamics of the pendulum in this region of the state space.

parameter α. As expected, we observe that for high values of α the controller obtained tends to stick to zero, while for small values of α its behavior is similar to previous cases when we did not take the norm of the control inputs into account.

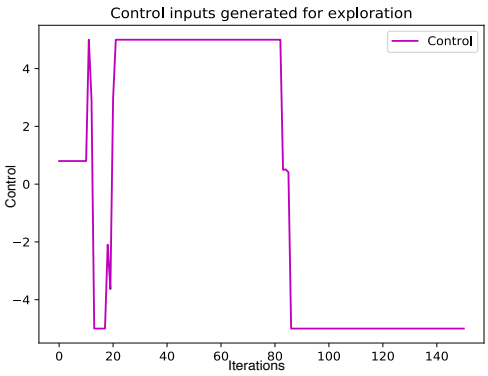
3.3 Conclusion on the greedy method

With this greedy method, the optimal control input is always chosen by looking for the most uncertain next state. However, it is still greedy: for complex nonlinear systems with bounded inputs, planning only one step ahead might not be enough to explore a large part of the state space. Instead, this control strategy will tend to oscillate between the regions of the input space that have not been explored yet, but never reach them, because each time it gets close to one, this region becomes less informative and it becomes more interesting to go to the next one. This sort of spiraling exploration strategy is suboptimal. Hence, we introduce a time horizon over which the control input is optimized, yielding more long-sighted strategies. The two main methods, presented in Chapter 4, plan several steps ahead, and explore the input space more efficiently.

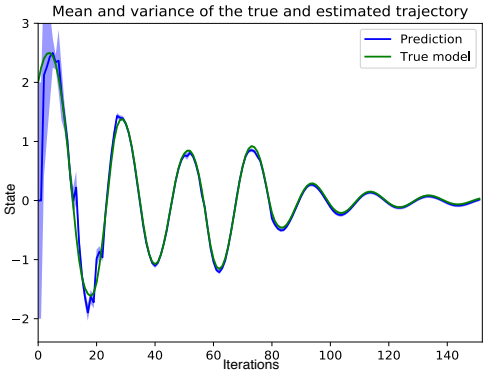
Note that if the control input is not bounded, then the system can go anywhere even just from one time step to the next,; in that case, the short and long-sighted strategies are equivalent. The active learning problem then reduces to a static problem, since no dynamics constrain it anymore, and classic methods for static GPs can be applied, such as those presented in Section 2.3.2. However, this is not the case here, as for the purpose of realism we assume bounded control inputs.



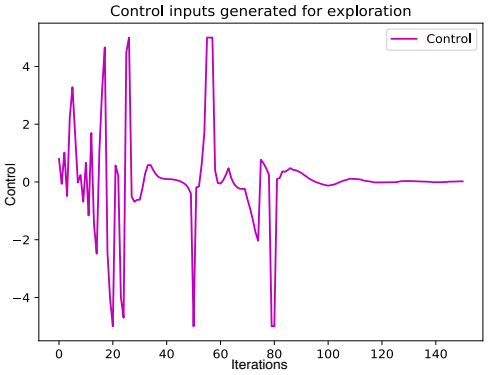
(A) True and estimated pendulum angle



(B) Control input when cost is posterior variance

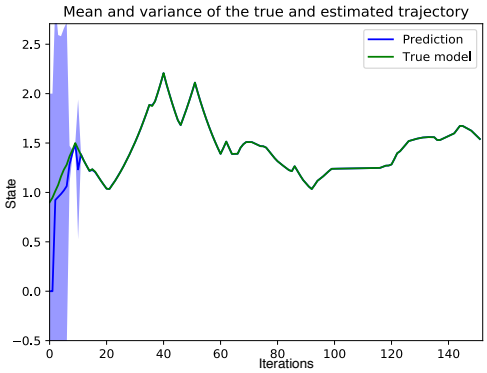


(C) True and estimated pendulum angle

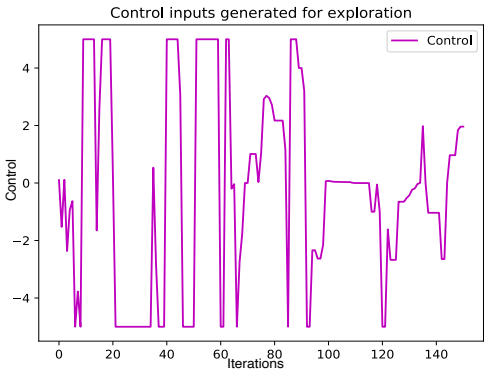


(D) Control input when cost is posterior variance and control norm, $\alpha = 1$

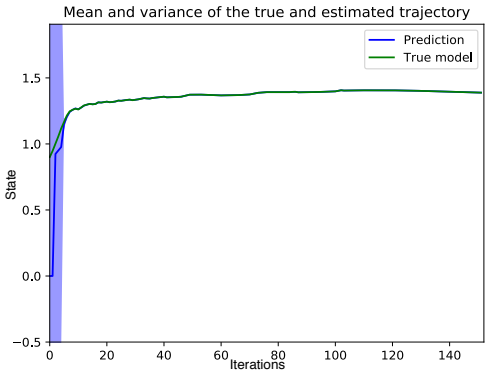
FIGURE 3.5: Greedy method with the pendulum simulations, either with bounded controller (top), or adding the control norm in the cost (bottom). We use the same level of process noise and $\hat{f}$ in the dynamics constraints, and $\alpha = 1$.



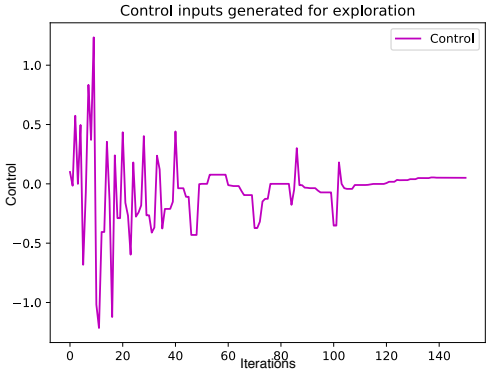
(A) True and estimated state x_2, added control norm with $\alpha = 0.1$



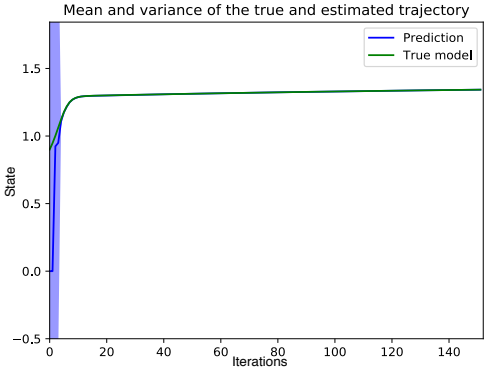
(B) Control input, added control norm with $\alpha = 10^{-2}$



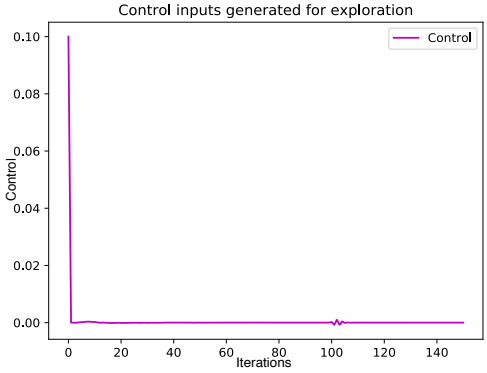
(C) True and estimated state x_2, added control norm with $\alpha = 1$



(D) Control input, added control norm with $\alpha = 1$



(E) True and estimated state x_2, added control norm with $\alpha = 100$



(F) Control input, added control norm with $\alpha = 10^4$

FIGURE 3.6: Greedy method with the CSTR simulations, when adding the control norm in the cost for different values of α.

Chapter 4

Two main methods and their theoretical analysis

In this chapter, we present our main methods for tackling the active learning problem (see Section 3.1 for the complete problem formulation). First, we design a separated search and control method, which is a direct extension of existing methods for the static sensor placement problem (see Section 2.3.2). This method greedily picks informative locations to visit, then designs a controller that drives the system there. Clearly, there are shortcomings to this as well, which we overcome with a novel method that optimizes the whole control trajectory over a fixed horizon. We start with a mathematical formulation of the optimal control problem, then present how it can be solved, and identify the main points to further investigate. We give a theoretical analysis of both proposed methods, then present empirical results on a numerical benchmark in the next chapter.

4.1 Separated search and control

We present a separated search and control approach for actively learning dynamical systems with GPs. This method is based on existing the state of the art for actively learning static GPs (see Section 2.3.2), which mainly rely on a property called submodularity. This property enables interesting results and theoretical guarantees; it would be beneficial if such results could be carried over to the dynamical setting. However, we show here that though the general framework can be extended for dynamical systems, providing similar theoretical guarantees is highly non trivial, and there are more efficient ways of approaching the problem.

4.1.1 Submodularity

We start by defining submodularity and the theoretical guarantees it can provide.

Definition 5. *Assume E is a finite and constant set. A set function $\phi\colon 2^E \to \mathbb{R}$, where 2^E is the power set, is submodular if $\forall\ A \subseteq B \subseteq E$, $e \in E \setminus B$, the following property holds:*

$$\phi(A \cup \{e\}) - \phi(A) \geq \phi(B \cup \{e\}) - \phi(B). \tag{4.1}$$

This property can be interpreted as diminishing returns, and is common in information theory. If a new element e is evaluated, it brings more information if the set of already known elements is small than if it is big.

Definition 6. *Assume E is a finite and constant set. A set function $\phi\colon 2^E \to \mathbb{R}$, where 2^E is the power set, is monotonic if $\forall\ A \subseteq E$, $e \in E$, the following property holds:*

$$\phi(A \cup \{e\}) \geq \phi(A). \tag{4.2}$$

A useful result by Nemhauser [34] follows:

Theorem 1 (Nemhauser's theorem, [34], proposition 4.3)**.** *If ϕ is submodular and monotonic such that $\phi(\emptyset) = 0$, then the greedy algorithm that starts with $A_0 = \emptyset$ and selects a sequence of sets*

$$A_{i+1} = A_i \cup \left\{ \underset{e \in E \setminus A_i}{\operatorname{argmax}}\, \phi(A_i \cup \{e\}) \right\}, \tag{4.3}$$

for $i \in \{1, ..., N\}$, with $N \in \mathbb{N}$ fixed, guarantees

$$\phi(A_N) \geq (1 - 1/e) \max_{|A| \leq N} \phi(A). \tag{4.4}$$

Equation 4.4 can be interepreted as follows: the sequence A_N of N elements selected according to (4.3) is at least as good as a constant fraction of the optimal sequence. This guarantees a certain level of optimality of the greedy sequence, which has the advantage of being relatively easy to compute, as opposed the true optimal sequence (NP-hard problem if ϕ is on one of the information criteria presented in 2.3.2).

Remark 1. *At each iteration of* (4.3)*, the next element $\{e\}$ needs to be chosen from the same set E for Theorem 1 to be applicable.*

In order to differentiate between iterations of the separated search and control procedure, and time steps in an experiment, we index the first ones with $i \in \mathbb{N}$ and name N the number of locations, and index the second ones with k and name n the number of data points.

The authors of [11] use Theorem 1 in the sensor placement problem to show that the sequence of sensor placements chosen by the greedy rule

$$x_{i+1} = \underset{x \in \mathcal{X}}{\operatorname{argmax}}\, I_i(x) \tag{4.5}$$

is suboptimal up to a fixed bound with respect to the exact solution. This holds for I any monotonic and submodular information criteria, such as the mutual information $I(X, f) = H(Y_X) - H(Y_X|f)$ between the observations Y made at X and the underlying function f, which we will use here. See [35], Section 3 and [9], Section II.B for more details. Since the optimal placement problem in that case is NP-hard and thus, the exact solution not feasible, this is a useful approximation which provides theoretical guarantees.

Notions from adaptive submodularity [36] enable similar results when the GP is updated and its hyperparameters are optimized after each iteration of (4.5), instead of being constant. The use of a greedy rule for maximizing a submodular information-theoretic criterion has produced interesting results on active learning for static GPs [11] [12] (or dynamical processes that can be modeled as such [13]). However, we will show in the next sections that extending this methodology to dynamical systems is highly non trivial.

4.1.2 Sketch of the approach

Next, we extend the static greedy rule (4.5) to dynamical systems and show the limitations of this approach. Contrarily to the static sensor placement problem, in a dynamical system we cannot decide to take a measurement at an arbitrary point x_i; we need to steer the system to x_i in order to collect data from this state. In order to manage these dynamical constraints, we propose a separated search and control approach: at each iteration, we first choose an informative location to visit in input space, then drive the system to this location. Hence, the dynamical nature of the problem is not explicitly taken into account in the exploration strategy.

We start from initial the state $z_0 := (x_0, u_0)$, and arbitrarily fix a number N of locations of interest to visit during the exploration procedure. At each iteration i of the greedy procedure, we pick the next state to visit according to the greedy rule:

$$z_i^G = \underset{x\in\mathcal{X},u\in\mathcal{U}}{\operatorname{argmax}}\, I(z \cup Z_{i-1}, f) = \underset{x\in\mathcal{X},u\in\mathcal{U}}{\operatorname{argmax}}\, H_{i-1}(z), \tag{4.6}$$

where $z = (x, u)$ is the complete GP input, $I(Z, f) = H(Z) - H(Z|f)$ is a monotonic, submodular information criterion and H_{i-1} is the differential entropy of the last GP, i.e., the GP when it was last updated (at iteration $i-1$). Note that it would also be possible to optimize directly over x_i^G instead of (x_i^G, u_i^G), however, this makes little difference in practice. Furthermore, optimizing for the complete GP input yields a more consistent interpretation with the other methods.

We then steer the system to this state x_i^G as efficiently as possible, by designing a control trajectory $(u_k, ..., u_{k+M-1}) \in \mathcal{U}^M$, where M is the control horizon, and k is the time step since the beginning of the experiment, while i indexes the iterations of the greedy procedure. Such a control trajectory exists for M large enough since we assume controllability. Once we are at the given location, we apply the control input u_i^G chosen by (4.6); we are then in the state $z_i^G \in \mathcal{X} \times \mathcal{U}$.

We solve this optimization with a simple NLP solver in CasADi. Since initialization is an important design choice in numerical optimization, we consider several possible initializations of the procedure. One of them, the so-called "continuous" initialization, consists in starting the optimization at the current location, so that a reasonably close local minimum will be returned by the greedy submodular procedure. It is also possible to evaluate the cost function at several random points around the current state and pick the one with the lowest cost ("random best" initialization). We can even run the whole optimization procedure from several random points around the current one and pick the one with the lowest final result ("random best optimum" initialization).

Once z_i^G has been returned by the optimizer, we need to design a controller that takes the system from the current state to x_i^G, where we will apply u_i^G. A good framework for this is the iterative linear quadratic regulator (iLQR), described in several papers [37] [38]. The linear quadratic regulator (LQR) operates for linear systems with a backward then a forward pass. Given a quadratic cost characterized by matrices Q and P, a fixed time horizon, an initial and a target state, the backward pass determines the sequence of optimal actions to take in terms of the state to reach at each step, i.e., optimal gains in a state-feedback form with an unknown sequence of states. The forward pass then starts from the known initial state, and replaces the states iteratively in the previous optimal sequence of actions. This algorithm can be adapted to nonlinear systems by locally linearizing the dynamics at each time step. We use a Python implementation of iLQR and compare several possible settings. We can either assume the true dynamics are known by the controller, i.e., the iLQR

controller actually takes the system to the state z_i^G. Or, in a more realistic setting, that the iLQR uses the estimated dynamics $\hat{f}$ to attempt to steer the system to z_i^G.

Note that using the true or estimated dynamics for control, the initialization, and the number N of locations to visit are important parameters for tuning this method. Next, we discuss its advantages and limitations.

4.1.3 Submodularity: from static to dynamic

We show in this section that the separated search and control method based on greedy rule (4.6) does not provide realistic theoretical guarantees.

Continuity of the input space As shown by Definition 5, submodularity is only defined for set functions, although some works exist on extending this concept to continuous functions. Hence, in order to use this property, $\mathcal{X} \times \mathcal{U}$ should be compact and discretized. This is not especially the case in this work, and not often for dynamical systems, which usually have continuous state spaces. However, this difficulty could be overcome (more or less artificially) by considering only finite input spaces.

Always picking the next element from the same pool In Figure 4.1, we represent the possible trajectories of a dynamical system as a tree, starting from a fixed state x_k. We name this tree of possible trajectories T_k. The most favorable problem formulation would have been to greedily select a whole trajectory through the tree by maximizing a submodular criterion, as in the following ideal formulation:

$$(X_k, U_k)^* = \underset{(X_k,U_k)\in T_k}{\operatorname{argmax}} \; I((X_k, U_k), f) = \underset{(X_k,U_k)\in T_k}{\operatorname{argmax}} \; H_k(X_k, U_k), \tag{4.7}$$

In that case, we would have been able to derive theoretical guarantees for the whole input trajectory (X_k^*, U_k^*) generated by the method, using Theorem 1. We would have been able show the sequence of trajectory chosen for exploration during one experiment is suboptimal up to a bound. This would have been an extension of the sensor placement results [11] that provides the same type of powerful guarantees.

However, this ideal extension does not hold, because formulation (4.7) does not fit into the category of problems defined by (4.3). As mentioned in Remark 1, the pool of elements from which the greedy rule of type (4.3) picks the next element should be constant for Theorem 1 to be applicable, but this is not the case here. Indeed, the tree T_k from which to choose the trajectory (X_k, U_k) changes at each time step, as for different values of x_k the root of the tree, different states are attainable in the tree. Hence, a problem of type (4.7) cannot provide any theoretical guarantees through the use of Theorem 1, and theoretical guarantees similar to those derived in [11] cannot be provided.

This shows that extending the active learning methods for static GPs [11] necessitates to choose the next sampling points from a constant pool of possible elements; and the only constant pool of inputs to choose from is the whole state space $\mathcal{X} \times \mathcal{U}$. With this in mind, we derive the greedy rule (4.6): we choose the next location to visit from the whole state space, hence, always from the same set of possible elements, which is conform to the assumptions of Theorem 1. We can then derive the following result:

Theorem 2. *Assume $\mathcal{X} \times \mathcal{U}$ is a constant, finite set of possible inputs. Assume each of the N locations $(z_i^G)_{i\in\{1,\dots,N\}}$ selected by* (4.6) *is actually the global maximum at that iteration.*

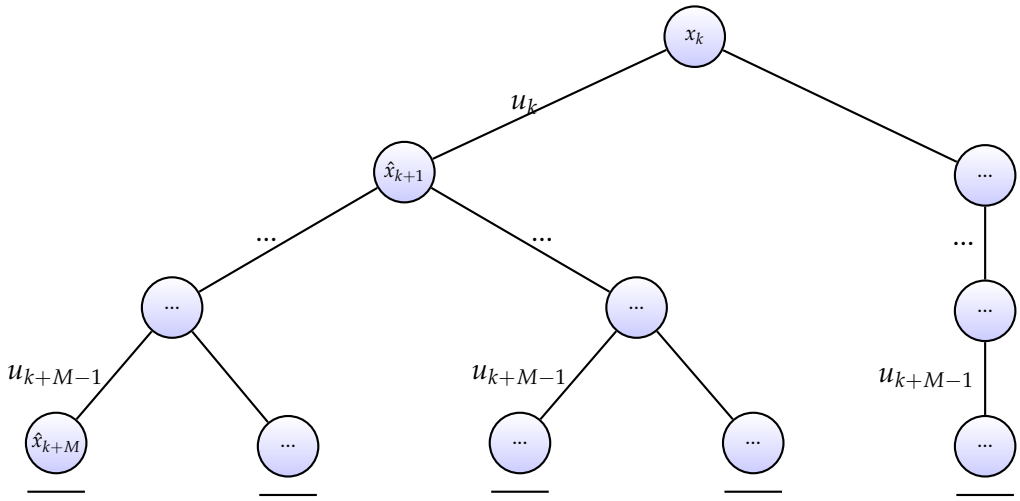


FIGURE 4.1: We represent the active learning problem as a tree T_k. Starting from the current position x_k, the first control input needs to be chosen, which takes the system to the next estimated state, etc for M steps. The aim is to generate informative control strategies $(u_k, ..., u_{k+M-1})$.

We denote

$$Z^G = \begin{pmatrix} z_1^G & \dots & z_N^G \end{pmatrix}^\intercal, \tag{4.8}$$

$$Z^{OPT} = \underset{z_1,\dots,z_N \; in(\mathcal{X}\times\mathcal{U})^N}{\operatorname{argmax}} I((z_1,\dots,z_N), f), \tag{4.9}$$

Then, the following guarantee on the sequence of locations selected by (4.6) *holds:*

$$I(Z^G, f) \geq (1 - 1/e) I(Z^{OPT}, f). \tag{4.10}$$

Here, $H(Z^G) = \frac{1}{2}\log|2\pi e K_{z^G}|$, *where* $K_{z^G} = (k(z_i^G, z_j^G))_{z_i^G, z_j^G \in (z_1^G,\dots,z_N^G)}$ *is the covariance matrix of the GP trained on observations of the sample points* $\begin{pmatrix} z_1^G & \dots & z_N^G \end{pmatrix}^\intercal$, *and* $I(Z, f) = H(Z) - H(Z|f)$.

Proof. Assume $\mathcal{X} \times \mathcal{U}$ is a constant, finite set of possible inputs. Assume each of the N locations $(z_i^G)_{i\in\{1,\dots,N\}}$ selected by (4.6) is actually the global maximum at that iteration. The function $I(Z, f)$ is then a submodular monotonic set function as of [9], [35]. Hence, the greedy rule (4.6) falls into the category of problems used in Theorem 1. A direct application of this theorem concludes the proof. □

This result guarantees $(z_i^G)_{i\in\{1,\dots,N\}}$ is a near-optimal sequence of locations to visit, in terms of maximizing its entropy. Note that it does not guarantee anything about the the trajectory in input space that is actually followed, or about the data gathered in between locations z_i^G.

Remark 2. *If we consider the GP is updated and its hyperparameters are optimized after each iteration i of* (4.6)*, a similar result can be obtained using adaptive submodularity [36].*

Never reaching the true greedy optimum However, we cannot guarantee the locations in Theorem 2 will actually be visited, hence, that (4.10) will hold in pracitce.

Indeed, z_i^G found at iteration i of (4.6) is the result of a non-convex, possibly high-dimensional optimization problem. Hence, it might not be the true maximum, even if a "random best" or "random best optimum" initialization is used. And even if it is the true maximum, we can only hope to approximately[1] reach it if the iLQR controller can use the true dynamics of the system and has enough time. If only the estimated dynamics $\hat{f}$ are available, which is the more realistic case, and if the time horizon of the iLQR is limited, then it is not very likely that z_i^G will actually be reached.

To sum up, it is highly unrealistic to assume that the state z_i^G chosen by (4.6) will actually be visited, and provide the data that the active sampling procedure chose to ask for. This shows that Theorem 2 does not hold in practice.

4.1.4 First tests

The separated search and control method still performs reasonably well, as we show in a few experiments. If we can design a good iLQR controller (artificially known dynamics for control), and if we have enough time between locations, then this strategy can be quite efficient. We call this the "artificially good" setting. In this setting, provided good initialization, the locations to visit push the system to regions far apart, which are then explored since we have accurate control.

In Figure 4.2, we compare the versions of this method in increasing order of realism: artificially good and unbounded controller, artificially good but bounded controller, and bounded controller with estimated dynamics. This is done for the pendulum simulations (see Section 5.1.1 for detailed dynamics), starting at $\theta = 0$, with horizon 50, $N = 6$, and matrices $G = 100 I_{d_x}$ and $P = 0.01 I_{d_u}$ used in the quadratic cost of the iLQR controller (see Section 4.1.2 for more details). We notice the rapid decrease in performance once only the estimated dynamics are used to design the iLQR that steers the system to each of the locations. This baseline can be powerful, but only if the corresponding controller performs well; a method that directly optimizes the control inputs, and therefore, takes the uncertainty on the control results into account, is often more efficient.

Conclusion on extending the submodular approach to dynamical systems In the end, we can say that it is highly non trivial to extend the state-of-the-art approaches for actively learning static GPs to dynamical systems. Indeed, though we were able to propose an approach based on similar ideas, similar suboptimality guarantees cannot realistically be provided. This is based on the fact that the pool from which to greedily choose the next sample to visit is often continuous, and changes due to the dynamics of the system; unless we can choose locations from the whole state space, in which case we will probably not reach them. Hence, we can conclude with the following revised theorem:

Theorem 3. *Procedure* (4.6) *does not verify the assumptions of Theorem 1 in general. Hence, Theorem 2 does not hold in practice.*

Proof. In general, $\mathcal{X} \times \mathcal{U}$ from which to choose the locations is continuous, and not a finite set. We also cannot guarantee that the locations $(z_i^G)_{i \in \{1,\dots,N\}}$ selected by (4.6) are actually optima. Theorem 2 does not hold without these two assumptions. And even if these assumptions were verified, we cannot guarantee that the locations (z_i^G)

[1] In any case, the iLQR is an approximate controller. For a complex nonlinear system, no matter which controller is used, reaching a reference target state is not trivial and cannot be guaranteed without exact knowledge of the dynamics.

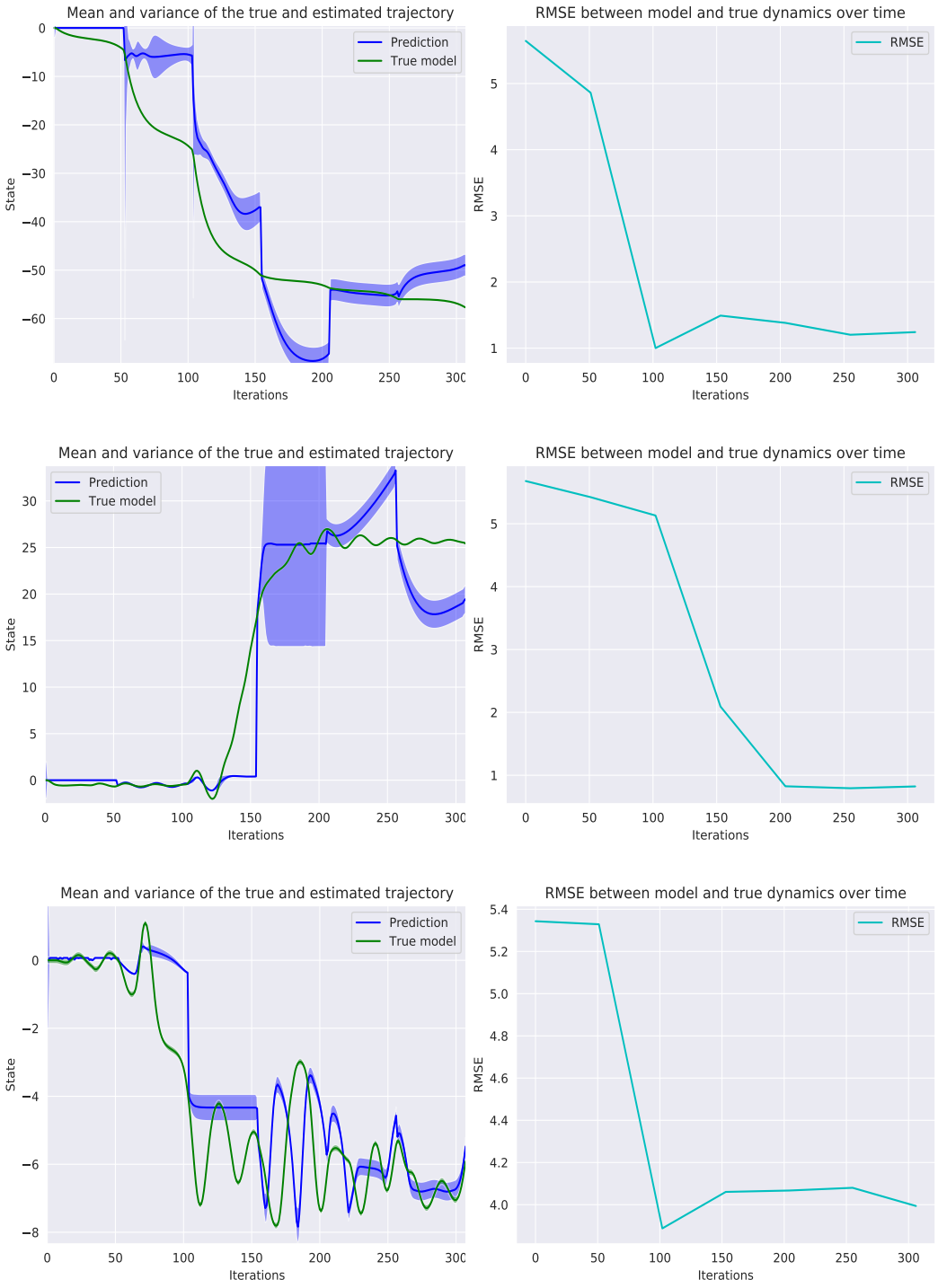


FIGURE 4.2: Three versions of the separated search and control method, with the pendulum simulations and $N = 6$: known dynamics with the unbounded controller (top), known dynamics with the bounded controller (middle), and finally bounded actuation with estimated dynamics (bottom). We show the true and estimated pendulum angle (left), and the error over time (RMSE, defined precisely in 5.1.3; right).

will actually be visited. This is even less likely if only the estimated system dynamics are available for controller design. Hence, suboptimality guarantees such as Theorem 2 do not hold in practice. □

Though it is an interesting extension of current active learning methods for static GPs, this separated search and control method cannot provide realistic suboptimality guarantees. On top of that, it is also does not optimize the data gathered along a whole trajectory, but only over a set of locations to visit. Indeed, Theorem 2 only bounds the suboptimality of the sequence of locations to visit, not the suboptimality of the whole trajectory in input space taken to visit them. Next, we will show that more efficient strategies can be designed by looking at the whole trajectory.

4.2 Optimal control for information maximization

The previous method chooses informative locations of the input space to visit, and then separately derives a controller to steer the system there. In the next sections, we present a method that uses the same information-theoretic criterion. This approach finds an informative control trajectory that optimizes it, while taking the dynamics constraints into account, showing superior performance.

4.2.1 Mathematical formulation

The greedy method presented in Chapter 3 shows that a non trivial strategy for exploring the state space can be derived from applying at each time step the control input that yields the next most uncertain estimated state, according to some information criterion. Returning to the original problem (3.7):

$$U_k^* = \underset{u_k,\dots,u_{k+M-1}\in\mathcal{U}^M}{\operatorname{argmin}} J((x_k,u_k),(\hat{f}(x_k,u_k),u_{k+1}),(\hat{f}(\hat{f}(x_k,u_k),u_{k+1}),u_{k+2}),\dots),$$

where $U_k^* = (u_k^*,\dots,u_{k+M-1}^*)$.

We focus on a natural criterion for the cost: we want to maximize the sum of the differential entropies of each estimated future state. At each iteration of the procedure, we pick the next most informative control trajectory by solving the optimization problem

$$\begin{aligned} U_k^* = \underset{u_k,\dots,u_{k+M-1}\in\mathcal{U}^M}{\operatorname{argmax}} & \sum_{i=0}^{M-1} H_k(\hat{x}_{k+i},u_{k+i}) \\ \text{s.t. } & \hat{x}_{k+1} = \hat{f}(\hat{x}_k,u_k),\ u_k\in\mathcal{U}\ \forall\, k, \end{aligned} \tag{4.11}$$

for a fixed time horizon $M \geq 1$, and $\hat{x}_k = y_k$ for $i = 0$ (simplifying notations). Numerically, the problem is solved using direct multiple shooting in the optimal control solver from CasADi[39] with Ipopt[40], as in [26] and [8] for example. CasADi is a symbolic optimal control solver, hence, it can leverage the fact that the GP's posterior mean and variance can be expressed analytically. Given a certain kernel with certain hyperparameters, we can then formulate the posterior mean and variance of $\hat{f}$ and the cost to be minimized in (4.11) as symbolic expressions in CasADi, which solves (4.11) using Ipopt. Since the problem considered is non convex and nonlinear, we can only hope to find local minima, and expect a large computational overload.

There are two possible settings for applying this optimal control method, which we denote "receding horizon" and "plan and apply".

Receding horizon The first possibility is to update the GP and optimize its hyperparameters at every time step, then recompute the whole trajectory, in a receding horizon, MPC-type of approach. This is simple and efficient: the larger M, the better (more long-term planning), and the GP is updated as often as possible no matter what. However, it is also very costly in terms of computations; by only updating the GP and control strategy every M time steps, i.e. rolling out the whole trajectory before updating everything again, the necessary resources are significantly reduced. Also, it can sometimes lead to a greedy behavior, since the exploration strategy has a chance to "change its mind" every time step, which will yield worse performance if the most informative path to choose is not clear.

Plan and apply Alternatively, we can roll out the whole control trajectory. At the end, after M steps, we have gotten to x_{k+M}, so we update the GP with the measurements taken along the way, optimize its hyperparameters, and iterate, in an compute-and-apply fashion. This setting necessitates fewer resources, but also comes with a trade-off. The largest M will not necessarily yield the best results, because a large M will not only mean long-term planning, but also waiting a long time before updating the learned model $\hat{f}$. Therefore, a value of M in between is best: large enough for planning, but not so large that the GP is not updated often enough.

Connection to Model-Predictive Control Model-Predictive Control (MPC) is a widely used tool for advanced control, which has produced large amounts of literature in recent years [41] [42]. In a classic optimal control problem, a cost is optimized while taking the dynamics into account as constraints, and the resulting controller ofhorizon M is applied to the system. Model-predictive controllers build on this approach, but instead apply only the first control input derived by the optimal control problem, then update the state of the system with the current measurements, and compute again the M next optimal control inputs. This type of receding horizon approach takes feedback from the system into account through new measurements and adjusts the controls as necessary, making it very efficient. It is also possible to include other constraints (such as terminal constraints) and complex costs in the MPC problem, making it a useful control tool, including for industrial settings.

The proposed optimal control approach (4.11) is similar to an MPC problem, hence, we can make use of the existing numerical tools for solving it, such as CasADi. It is also as versatile as an MPC formulation, since other costs or constraints can be included for practical applications. But it optimizes an information-theoretical cost instead of a control one, and only has access to an estimate $\hat{f}$ of the true dynamics f. Since it is similar to an MPC formulation, it also has some of its drawbacks, such as high computational burden and sensitivity to the model quality. Note that similar control approaches have been used in some existing works for exploring dynamical systems [43] [26], but in different settings, focusing more on linear systems or on safety for example.

4.2.2 Influence of the planning horizon

In Figure 4.3, we can see the obtained trajectory for the pendulum simulation with different values of the time horizon M. The estimated (blue) and true (green) angle of the pendulum are plotted along with the entropy of the GP at the next estimated point, showing that with longer horizon, the system explores regions with high entropy faster and goes further away from the initial position faster. For example, with $\mathcal{U} = [-5; 5]$, the short-sighted solution jumps to quickly from $-u_{\max}$ to $u_{\max}$,

and can therefore, only generate small oscillations that can even slowly damp out. On the contrary, longer time horizons are able to learn how to use previous oscillations to get the pendulum much higher, and perform a swing-up, after about 70 iterations for $M = 5$ and 50 iterations for $M = 20$, which takes them to unknown regions of the state-space.

4.2.3 Adding other costs

As we did for the greedy method in Section 3.2.2, we investigate the possibility of adding extra terms to the cost function used in (4.11). As a proof of concept that this is possible with our method, we propose the following variant of (4.11):

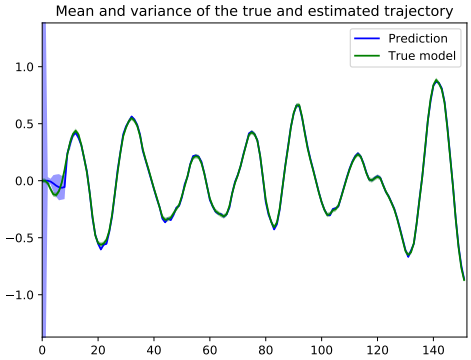
$$\begin{aligned} U_k^* = & \underset{u_k,\dots,u_{k+M-1}\in\mathcal{U}^M}{\operatorname{argmax}} \sum_{i=0}^{M-1} H_k(\hat{x}_{k+i}, u_{k+i}) + \alpha||u_k||_{L^2} \\ & \text{s.t. } \hat{x}_{k+1} = \hat{f}(\hat{x}_k, u_k),\ u_k \in \mathcal{U}\ \forall\ t. \end{aligned} \tag{4.12}$$

Adding $\alpha||u_k||_{L^2}$ in the optimization problem reduces the control costs and can also serve as a type of regularization. The generated control inputs are smoother and less aggressive, which is often necessary for industrial systems in order to avoid damage. However, it also reduces the amplitude of the control signals generated, making exploration less efficient.

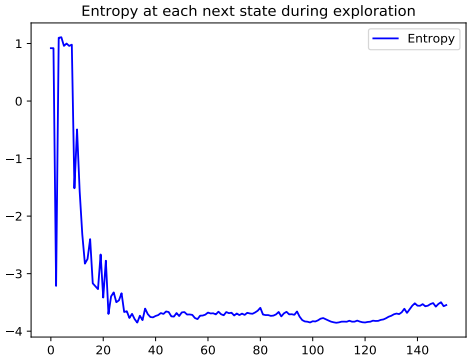
In the end, we decide not to add this extra term in our optimization procedure, as in our practical examples the observed behaviors were efficient and diverse enough. Also, with the hyperpriors we choose in order to robustify the learning procedure (see Section 5.1.4), it does not seem necessary to regularize for the training to converge well. But we have conducted several experiments using this extra cost, even if our final simulations do not; this serves as a proof of concept that other criteria can be taken into account by our method. It is one of the advantages of this optimal control method that it can take other criteria into account than just information maximization, not only control costs but any type of other behavior, such as staying close to a reference trajectory for example. Hence, this method not only performs well in terms of exploration, but also enables other aspects to be taken into account.

4.2.4 Advantages and limitations of the method

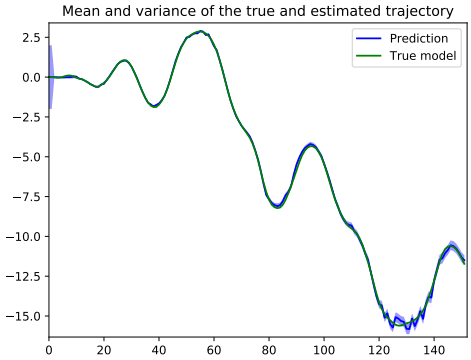
This method based on optimal control generates informative control trajectories that steer the system to regions of high uncertainty in the input state. Because of its very general formulation, it is highly versatile: we have observed aggressive controls resembling bang-bang where u_k mostly took values $\pm u_{\max}$, but also much more complex behaviors for systems for which this was less optimal. It also enables taking other considerations into account, simply by adding extra terms to the information criterion in the optimal control problem, as shown in Section 4.2.3. This strategy directly uses the model $\hat{f}$, and takes the dynamics into account as constraint, while optimizing over the whole trajectory in the next M time steps. All model-based active learning strategies (e.g., all those we propose in this thesis, as opposed to random approaches such as standard system identification signals) explicitly explore both $\mathcal{X}$ and $\mathcal{U}$. But the fact that they directly depend on the GP model can also be considered a risk: if $\hat{f}$ is very much off, for example if the hyperparameters are badly initialized and do not converge with the available data, then the exploration strategy might not generate useful control inputs. Therefore, providing good priors for the hyperparameters can be essential to the success of this method. For example,



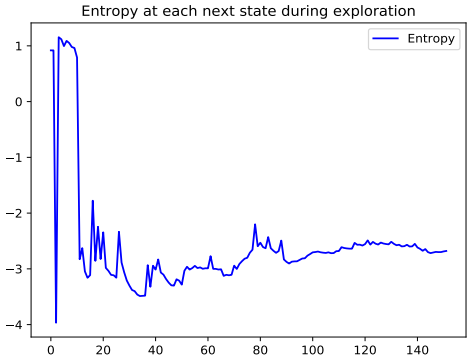
(A) True and estimated pendulum angle, $M = 1$



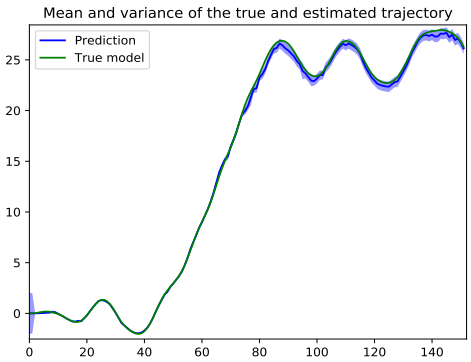
(B) Entropy of the GP at the next estimated point



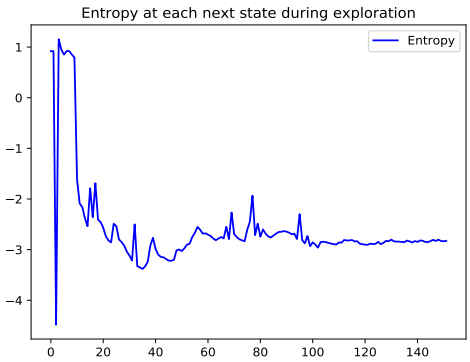
(C) True and estimated pendulum angle, $M = 5$



(D) Entropy of the GP at the next estimated point



(E) True and estimated pendulum angle, $M = 20$



(F) Entropy of the GP at the next estimated point

FIGURE 4.3: Comparison of different time horizons for the exploration strategy with the pendulum simulations. We plot the estimated pendulum angle (blue) along with the true value (green). We also show the entropy from the GP at each estimated next point, showing that strategies with longer horizons explore regions of higher entropy faster and go further away from the initial position sooner, learning to perform a first swing-up after about 70 iterations for $M = 5$ and 50 iterations for $M = 20$. Note that with $M = 20$, the pendulum also swings much more times than with $M = 5$, yielding a higher cumulated angle.

a useful approach for practical applications could be to start generating data with one of the random exploration methods, derive reasonable hyperparameters from this initial data, then start using (4.11) with Gaussian priors over these values of the hyperparameters.

4.3 Event-triggered switch between exploration and exploitation

As described at the beginning of Section 4.2.1, there are two ways the optimal control method can be applied: either in a receding horizon, or in a plan and apply fashion. Having a large horizon M and updating the GP every time step (receding horizon) is often more advantageous, since we can plan further ahead while still learning from the generated data often. However, this also imposes a large computational overhead, and can sometimes even lead to a greedy behavior is the strategy "changes its mind" too often. Hence, the plan and apply formulation, which shows surprisingly similar performance in experiments, is promising.

We propose to find a compromise between these two variants, by distinguishing between two possible cases. On the one hand, the GP has learned well and the model is accurate, hence, we can plan several steps ahead and avoid wasting computational power on updating it all too often. On the other hand, the model is not accurate, hence, model uncertainty make it impossible to predict far into the future, and the GP needs to be updated. In the following sections, we design two event-triggered switches that leverage these two types of behaviors.

4.3.1 Greedy switch

We distinguish two behaviors:

- Exploration: greedy, local exploration of a certain region, until the GP has gathered enough information about this region and has adapted its hyperparameters, and gives good predictions in this region. In this phase, the active learning strategy chooses the control trajectory as

$$u_k^* = \underset{u_k}{\operatorname{argmax}}\, H_k(x_k, u_k), \tag{4.13}$$

  i.e. greedy strategy with $M = 1$. That way, when we are in a new region of the input space, we stay there and update the GP very often; we switch to the other phase once we have learned this region well enough.

- Exploitation: long-term exploration, exploiting the local knowledge we have gathered until now to plan long-term and reach new regions of the input space. In this phase, the active learning strategy chooses the control trajectory as

$$\begin{aligned} U_k^* = \underset{u_k,\dots,u_{k+M-1}\in\mathcal{U}^M}{\operatorname{argmax}} & \sum_{i=0}^{M-1} H_k(\hat{x}_{k+i}, u_{k+i}) \\ \text{s.t. } \hat{x}_{k+1} = \hat{f}(\hat{x}_k, u_k),\ & u_k \in \mathcal{U}\ \forall\, t, \end{aligned} \tag{4.14}$$

  with a large value of M to be specified later. That way, once we have learned a certain region of the input space well, we can plan several steps ahead in order to get out of this local bassin of attraction and reach a new region, in which we will then switch back to the exploration phase.

We switch from the exploration to the exploitation phase as soon as a local region has been learned well, and back to exploration as soon as we reach a new region on which the GP performs poorly. We propose the following trigger for switching between the two types of behavior:

$$S(k,P) = \frac{1}{P}\sum_{i=0}^{P-1} ||y_{k-i} - x_{k-i}^{\text{estim}}||_{L^2}, \tag{4.15}$$

where y_k is the (noisy) measurement of x_k given to the GP for training, and x_{k-i}^{estim} was the value of x_{k-i} estimated by the GP at time $k-i-1$, i.e. $x_{k-i}^{\text{estim}} = \hat{f}_{k-i-1}(x_{k-i-1}, u_{k-i-1})$. Hence, no new computations need to be conducted, we simply evaluate the average prediction error over the last few data points; if this error was low, the model is accurate and we can plan long-term, otherwise we need to update it. With η a threshold set by the user, which characterizes the tolerance of the user in terms of local prediction error, we define the switch:

- If $S(k,P) \geq \eta$, switch to the exploration phase;
- If $S(k,P) \leq \eta$, switch to the exploitation phase.

Choice of threshold η There exists an upper bound on η over which the system will always be exploiting, therefore, the GP might not converge well. Similarly, there is a lower bound under which the system will never exploit and therefore, stay greedy. For the pendulum with $\sigma_\epsilon^2 = 0.05$, these bounds seem to be around 0.2 and 0.4. We provide the user with some heuristics on the interval in which η can be chosen, and a risk-seeking user can then select a higher value from this interval (more exploitation), a safety-conscious user a lower value (more local exploration).

The threshold η is naturally bounded: we have $y_{k-i} = x_{k-i} + \epsilon_{k-i}$, where $\epsilon_{k-i} \sim \mathcal{N}(0, \sigma_\epsilon^2 I_{d_x})$. In order to avoid mistaking measurement noise for learning error, we need η to be larger than ϵ_{k-i} in 99.7% of cases, and therefore, derive the following bound from simple Gaussian tail bounds:

$$\eta \geq 3\sigma_\epsilon. \tag{4.16}$$

Similarly, a good rule of thumb can be derived for an upper bound on η. If we consider σ_{prior} the prior variance of the GP, set by the user before any measurements are taken, then σ_{prior} can be interpreted as the variance that the user expects from the prior model. It would therefore, make little sense to say that even if the current model is much more wrong than the prior, the GP should keep exploiting instead of updating locally until the error goes down. Hence, forcing exploitation even when η is outside of the expectations of the prior model, i.e. $\eta \geq 3\sigma_{prior}$ for a Gaussian prior, would make little sense. Therefore, we can derive a heuristic upper bound:

$$\eta \leq 3\sigma_{prior}. \tag{4.17}$$

In our experiments, we did some tuning by hand to find the value of η that best fit our purposes for each system we studied. With $\sigma_\epsilon = 0.05$ and $\sigma_{prior} = 1$ (standard Gaussian prior), the values of η that seemed reasonable to us were always inside those heuristic bounds.

Choice of P If the user can estimate the regularity of the dynamics f, a good choice for P could be $P = \lceil L \rceil$, where L is the Lipschitz constant of f. For the pendulum,

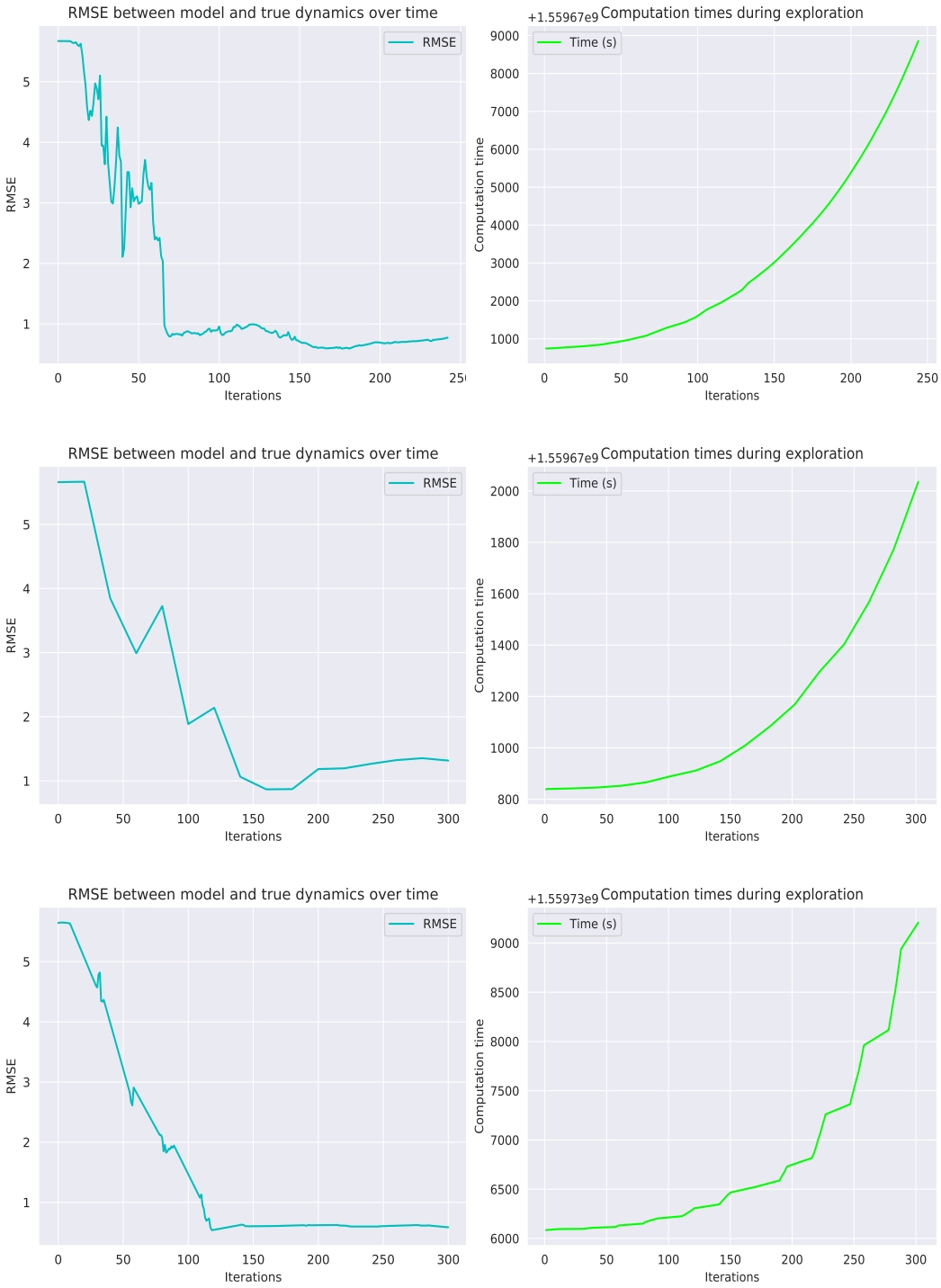


FIGURE 4.4: Compare exploration trajectory, RMSE and computation time for three versions of the optimal control method: planning $M = 20$ steps ahead and updating both control and GP every iteration (receding-horizon, top), planning M steps ahead and updating both only every M steps (plan and apply, middle), and using the greedy switch to choose between $M = 1$ and $M = 20$ after each iteration (bottom).

with our parameters, we have $P = 14$ with that strategy, which sounds coherent compared to the experimental data. For a same value of M, choosing P too small can change the results (for $\eta = 0.5$, $M = 20$ and the pendulum simulation, $P = 10$ gave good results but not $P = 5$).

Otherwise, a good heuristic seems to be choosing P close to $M/2$, which is what we did for our experiments.

Choice of M for exploitation Due to the nature of the time horizon, we have $M \geq 1$. Note also that CasADi, in order for the symbolic optimization procedure to run, creates a large graph representing the cost function for direct single or multiple shooting, which grows exponentially in M. Hence, in order to keep both the computation time and the memory requirements from CasADi reasonable for a regular desktop computer, we only consider $M \leq 20$. We usually test several values for this horizon, and find that $M = 10$ or $M = 15$ yield good performance (planning horizon long enough for exploration, but GP updated often enough).

Experimental tests We start by implementing the switch with our previous pendulum simulations (see Section 5.1.1 for detailed dynamics). Recall that the aim of this event-triggered switching between exploration and exploitation lies in finding a good balance between computational burden, convergence of the GP and an efficient exploration. We aim at updating the GP as soon as it is needed (exploration) in order to gather enough data and learn the whole system well. On the other hand, we avoid unnecessary calculations such as updating the GP too often if the model is already accurate, or planning a long horizon ahead when there are very informative points close by.

This trade-off is illustrated in Figure 4.4. We compare the exploration trajectory, error (RMSE, defined precisely in 5.1.3) over time and computation time. In the top figure, the receding horizon setting is used. In the middle figure, we plan the control trajectory and update the GP every M steps. In the bottom figure, we use the greedy switch described in this section. The experiments were run with $M = 20$, $\eta = 0.5$ and $P = 10$. The first solution shows very promising exploration results, with the lowest final error, learning to do a swing-up early on, but also the highest computational cost. On the other hand, the second strategy yields a somewhat higher error and less efficient exploration, but much less computation. The switching strategy is then able to balance both aspects, reaching an error almost as low as the receding horizon variant, with less than half as much computation time. It leans more to one variant or the other depending on the parameters chosen, and grows greedier as η gets smaller.

4.3.2 Receding-horizon switch

The previous idea for managing exploration and exploitation in an event-triggered way switches between greedy, local exploration and long-term exploitation. In order to leverage the high performance of receding horizon while limiting the computational costs, we propose another approach: always planning long-term, but adapting the rate at which the GP is updated. This boils down to switching between the "receding-horizon" and "plan and apply" versions of the optimal control method (see

4.2.1). In both phases, the active learning strategy chooses the control trajectory as

$$U_k^* = \underset{u_k,\dots,u_{k+M-1}\in\mathcal{U}^M}{\operatorname{argmax}} \sum_{i=0}^{M-1} H_k(\hat{x}_{k+i}, u_{k+i})$$
$$\text{s.t. } \hat{x}_{k+1} = \hat{f}(\hat{x}_k, u_k),\ u_k \in \mathcal{U}\ \forall\ t, \tag{4.18}$$

but adapts the updating rate of the GP. This yields the following behavior:

- Exploration: long-term exploration (4.18), updating the GP at each iteration.
- Exploitation: long-term exploration (4.18), updating the GP only every M steps.

The switching behavior is the same as previously, with $S(t, P)$ as before:

- If $S(t, P) \leq \eta$, switch to the exploitation phase;
- If $S(t, P) \geq \eta$, switch to the exploration phase.

This switch yields higher computational costs than the baseline switch, but also higher performance. The previous sections concerning the choice of parameters for the greedy switch also apply for this one.

4.4 Conclusion on the main methods

In the previous chapters, we presented the problem of actively learning dynamical systems with GPs, and have discussed some of the existing works in this field. We have proposed several methods, starting with a greedy method (Section 3.2), then a separated search and control method extended from the literature concerning active learning with static GPs (Section 4.1), and finally an informative control generation method based on optimal control (Section 4.2). We have discussed the advantages and drawbacks of each approach theoretically. The proposed methods and associated parameters are summed up in Table 4.1. In the next chapter, we benchmark these different ideas on a set of numerical examples, and draw conclusions on their performance.

TABLE 4.1: Summary of the proposed methods, and their main parameters and characteristics. The computational bottleneck of each method is indicated in the computation column.

Method	**Main parameters**				
	Model-based	Planning horizon	Receding time	Computation time	Important parameters
Standard system identification signals	No	No planning	No	Very short (update GP)	Holding time
Greedy	Yes	$M = 1$	No	Long (update GP every step)	Cost function
Seperated search and control	Yes	Time between locations	No	Long if true dynamics (iLQR)	Cost, N, true or estimated dynamics for control
Optimal control, receding horizon	Yes	M	Yes	Long (plan and update GP every step)	Cost, M
Optimal control, plan and apply	Yes	M	No	Rather short (plan and update GP every M steps)	Cost, M
Greedy switch	Yes	M	No	Rather short (greedy phases)	Cost, M, η (switch threshold)
Receding horizon switch	Yes	M	Yes if activated	Rather long (receding horizon phases)	Cost, M, η (switch threshold)

Chapter 5

Experimental results

In this chapter, we evaluate the methods proposed in this thesis on numerical experiments. First, we present the set-up of our benchmark, then the main experimental results. The different methods we compare are summarized in Table 4.1. We start by listing the benchmark systems and their parameters. We then discuss the experimental conditions and choices, along with some problems that were faced, before presenting the experimental findings.

5.1 Experimental set-up

First, we describe the experimental set-up used for this benchmark. We present the systems used for testing our methods: an inverted pendulum, a continuously-stirred tank reactor, a two-link robot manipulator, a double inverted pendulum on a cart, a unicycle, and the half cheetah robot model. The discrete-time dynamics are solved numerically, and the methods proposed in this thesis are applied to all systems. Their results are then compared using several metrics, such as the root mean squared prediction error over an evaluation grid, and the percentage of that grid that was actually visited by the system.

5.1.1 Dynamics tested

We test the proposed methods on different dynamical systems. Their dynamics are described in the following section; in each case, we solve the differential equations describing them explicitly with the Runge-Kutta 4/5 method [32]. We select appropriate parameters for all the benchmark systems by trial and error. Indeed, we need to tune them until they enable a certain differentiation between the proposed methods. If a system is very easy to actuate, for example because it is very lightly damped or because $\mathcal{U}$ is very large, then all methods will be able to generate controls that explore the whole input space efficiently. On the other hand, if it is very hard to actuate, then none of the methods will manage to explore it. We aim at finding mostly systems that lie in between, i.e., which are feasible for some methods and not for others, in order to demonstrate the difference between them. The chosen parameters still remain realistic: we end up mostly choosing the damping of the system so it is physically realistic and not all too easy to actuate, then limiting $\mathcal{U}$ so that not all methods can explore the system as efficiently.

Pendulum We implement the nonlinear dynamics of an inverted pendulum. This is a standard benchmark in control theory on which to try our algorithms. Also, the dynamics are well-known and easily visualizable, which makes it convenient to

interpret. We use the state $x = (\theta, \dot{\theta})^\intercal$ and the dynamics

$$\dot{x} = \begin{pmatrix} \dot{\theta} \\ -\frac{g}{l}\sin(theta) - \frac{k}{m}\dot{\theta} - u \end{pmatrix}, \tag{5.1}$$

with $g = 9.8$ the gravity constant, $m = 0.1$ the mass of the pendulum, $l = 1$ its length, and $k = 0.05$ the damping factor. We use $\mathcal{U} = [-5; 5]$ in the early experiments, $\mathcal{U} = [-3.5; 3.5]$ in the final benchmark.

CSTR The continuously-stirred tank reactor (CSTR) is an example often used in the MPC literature [33]. Its dynamics are highly nonlinear and unstable, therefore, not easy enough to learn to be very illustrative of our work; some results with this system are shown in Chapter 4. Hence, we do not include it in the final benchmark, but still present its dynamics:

$$\dot{x} = \begin{pmatrix} \frac{1}{\theta}(1 - x_1) - kx_1 e^{-\frac{M}{x_2}} \\ \frac{1}{\theta}(x_f - x_2) + kx_1 e^{-\frac{M}{x_2}} - u(x_2 - x_c) \end{pmatrix}, \tag{5.2}$$

with $x = (x_1, x_2)^\intercal$, the parameters used in [33], and $\mathcal{U} = [-3; 3]$. This system is highly unstable and not controllable everywhere, therefore, it is not the best system to test our methods on, and we do not further exploit it.

Two-link robot Two-link planar manipulators are classical examples in robotics; they are well-studied, stable, and already show complex behavior. Another advantage of testing our framework on these systems is that they scale easily, i.e., it is fairly straightforward to go from a two-link, rather simple robotic arm to multiple links for a more complex robotic system. We use the states $x = (\theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2})^\intercal$, with $F = (u_1, u_2)^\intercal$ the torques imposed by the controller, and the dynamics

$$\dot{x} \quad = \begin{pmatrix} \dot{\theta_1} \\ \dot{\theta_2} \\ B(\theta)^{-1}(F - C(\dot{\theta}, \theta) - h(\theta) - F_v\dot{\theta} - F_s\,\mathrm{sign}(\dot{\theta})) \end{pmatrix}. \tag{5.3}$$

We set the parameters as follows: $g = 9.8$ the gravity constant, $m_1 = m_2 = 1$ the mass of each arm, $l_1 = l_2 = 1$ their length, $r_1 = r_2 = 1/2$, and

$$I_i = \begin{pmatrix} \frac{1}{3}m_i l_i^2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3}m_i l_i^2 \end{pmatrix} \tag{5.4}$$

the inertia matrix of each arm. The system is illustrated in Figure 5.1.

Considering the system without damping leads to serious divergence issues, and is not very realistic from an application driven point of view. Hence, we add damping. This is also beneficial for our algorithms, since the system is then stable, and exploring it becomes less easy for naive methods. After some trials on the simulated system, we pick the viscous and static friction matrices $F_v = F_s = \begin{pmatrix} 0.4 & 0 \\ 0 & 0.4 \end{pmatrix}$, and use $\mathcal{U} = [-5; 5]$.

Unicycle The unicycle is a nonholonomic system, hence, interesting to test our different active learning strategies on. Indeed, its dynamics are constrained: not all

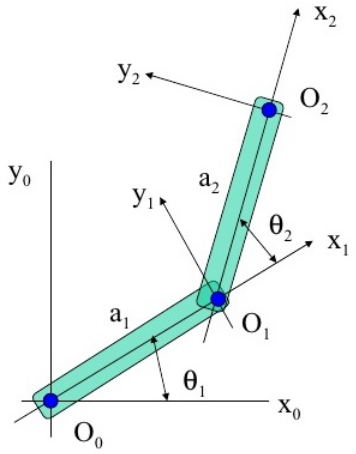


FIGURE 5.1: Illustration of the two-link robot manipulator. Source: https://www.slideshare.net.

movements are possible, and the control inputs have more or less effect depending on the current state of the system (the forward thrust is the projection of the forward control onto the current forward axis of the system). Hence, actuating the unicycle and exploring its state space is non trivial. Its dynamics can be written as follows, with states $X = (x, y, \theta, \dot{x}, \dot{y}, \dot{\theta})^\intercal$, and imposed torques $T = (u_1, u_2)^\intercal$:

$$\dot{X} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ M^{-1}F + M^{-1}A^\intercal(AM^{-1}A^\intercal)^{-1}(b - AM^{-1}F) \end{pmatrix}, \tag{5.5}$$

where

$$M = \begin{pmatrix} m & 0 & -mr\sin(\theta) \\ 0 & m & mr\cos(\theta) \\ -mr\sin(\theta) & mr\cos(\theta) & I_c \end{pmatrix} \tag{5.6}$$

is the inertia matrix of the unicycle, with m its mass, r the length between center of mass and center of rotation, θ the angle formed by its forward direction, and I_c its moment of inertia about the center of mass.

$$F = \begin{pmatrix} u_1\cos(\theta) + mr\dot{\theta}^2\cos(\theta) \\ u_1\sin(\theta) + mr\dot{\theta}^2\sin(\theta) \\ u_2 \end{pmatrix} \tag{5.7}$$

are the forces applied;

$$A = \begin{pmatrix} \tan(\theta) & -1 & 0 \end{pmatrix} \tag{5.8}$$

and $b = -\dot{x}\dot{\theta}\sec(\theta)^2$ are the terms describing the constraints. We use $m = 1$, $r = 0.05$, $I_c = (\frac{m}{2})^2$, a damping coefficient of 1, and $\mathcal{U} = [-0.05; 0.05]$. The system is illustrated in Figure 5.2.

OpenAI Gym systems OpenAI's Gym package provides benchmarks for reinforcement learning, including robots from MuJoCo. MuJoCo is a physical simulation

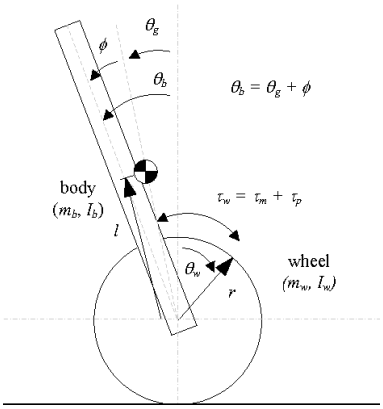


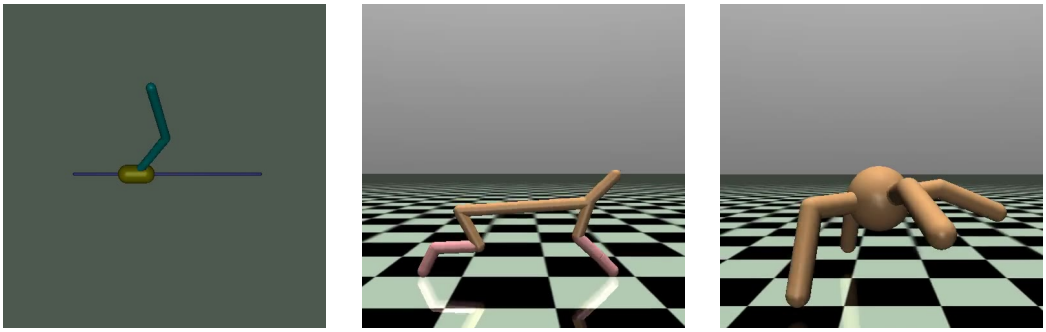
FIGURE 5.2: Illustration of the unicycle. Source: https://www.semanticscholar.org/.

environment often used for benchmarking methods in reinforcement learning, including many works in legged locomotion for example. Interesting systems for us include the ant, the half-cheetah, the (double) inverted pendulum on a cart and the robot manipulator. Because of its presence in many previous papers [29], [44], [45], using Gym makes for more reproducible and comparable experiments. We focus on the tasks with continuous action state, and therefore, exclude simple RL tasks such as the discrete car on a mountain or Atari games. Since we only need symbolic expressions for the posterior mean and variance of the GP, the Gym systems can be integrated into our previous framework, by simply keeping track of the environment of a system as a parameter of our method and sequentially applying to it the controls selected by the active learning procedure. We focus on the MuJoCo environments, which include interesting systems with rather realistic dynamics for our use cases.

Note that the Gym systems often model angles by $\cos(\theta)$ and $\sin(\theta)$, while we modeled them by θ directly in the dynamics we implemented. This has advantages and drawbacks. On the one hand, using the angle directly is closer to the physical reality, and makes the notion of exploration and of visiting new regions more explicit. Also, learning $\cos(\theta)$ and $\sin(\theta)$ independently means that the GP is not aware that they are correlated, hence, loses a lot of information coming from the physical reality of the angular measurement. On the other hand, this also means that the value of the angle will get huge, and that the GP has to learn from scratch again at each new region, since the positions at $\pm 2\pi$, which are actually the same, are not spatially correlated, and therefore, not perceived as having the same behavior for the GP (since there is different behavior in between). Another possibility would be to use a periodic kernel with the real angle, which encompasses the periodicity of the data directly into the kernel. However, using a naive implementation of this kernel can lead to numerical instabilities. On top of this, for most system only some of the states are periodic, hence, we would need to have different kernels for different dimensions, which is less general and requires knowledge about the system. In the end, we decide to keep a squared exponential kernel and learn $\cos(\theta)$ and $\sin(\theta)$ for the double inverted pendulum on a cart (DIPC) from OpenAI Gym, but keep the real angle for the systems we implemented ourselves. It is interesting to note however, that such design choices also have an impact on the learning procedure and therefore, on the model-based exploration.

In order to benchmark our methods on the Gym systems, we decide to:

- Select only some of the MuJoCo systems, which have more or less realistic dynamics and are mostly stable (though sometimes not strongly damped).
- Select which observations to use: instead of using the observations provided by Gym, we can use the internal MuJoCo states directly, that have not been transformed by Gym. That way, we observe the angles directly instead of their sine and cosine, the velocities are not clamped, and the dynamics are overall closer to the true physical dynamics than when using the observations provided by Gym. However, this is not always a good idea; in the case of the double pendulum on a cart for example, we end up using the transformed states in Gym instead of the ones used in MuJoCo, because learning $\cos(\theta)$ and $\sin(\theta)$ seemed easier for the GP than learning directly θ.
- Set the state directly: Gym only allows `reset()` to set the state of the system, in a way that is defined differently for each system (either start at a defined initial state, or at a random state). However, in order to benchmark our methods, we want to start from the same equilibrium point at each trial, which is not the case with most of the Gym initializations. We also need to be able to set the simulation state for evaluation: at each point x in the evaluation grid, we need to set the simulation state to x, apply a random control and observe the state x_{next}. Setting the state can be done by accessing directly the internal MuJoCo states. Keep in mind that when using undirect observations, i.e., the Gym observations directly, it will be necessary to return to the internal MuJoCo states when setting the Gym environment, therefore, the transformation from MuJoCo to Gym states should be invertible and its inverse used each time we set the environment if undirect observations are used. Hence, we usually use the direct observations by setting our state x to the internal MuJoCo state.
- Select the grid on which to evaluate the learned model carefully. If the grid is too large, for example if it contains points for which the simulation or the system itself are unstable, then they might not be useful for evaluating the current model. But if it is too narrow, then it will not differentiate as much as possible between the exploration strategies, since strategies that stay in one region of the state space instead of exploring it all will be rewarded. The user should make sure that the chosen grid only encompasses points that are feasible for the system; MuJoCo will not always raise an error when physically impossible states are simulated, but often produce bogus predictions.
- Start at a stable equilibrium, and select $\mathcal{U}$ carefully. If the control input can be very large, and/or the initial state is already an interesting and hard to access region of the state space, for example in the case of the double pendulum on a cart (can push the cart up to $1m/s$, and starts with an upper vertical pendulum in Gym), then naive exploration strategies such as random bang bang are already able to explore all of the state space, since no long-term planning is necessary to visit regions that could be hard to access. By starting from a stable equilibrium and using a reasonably limited control input, we place ourselves in a more realistic context, and also choose problems that are illustrative of the advantage of model-based exploration methods compared to naive ones, since they are too hard for exploration to be possible without planning, but feasible at various degrees with our exploration schemes.



(A) DIPC (B) Half-Cheetah (C) Ant

FIGURE 5.3: Illustration of the Gym systems we use. Source: https://gym.openai.com.

5.1.2 CasADi, Ipopt, and the code

The code used for the simulations is available on git. We start with an initial state, and initial controller, and a standard Gaussian prior of mean 0 and variance 1. For the initialization, we compute $x_1 = f(x_0, u_0)$ and estimate $\hat{x}_1$ again with a mean of 0 and a variance of 1. After this initial guess, we create the GP model. At each iteration of a model-based active learning procedure, we call CasADi [39] to find the next optimal control sequence. This means that instead of finding the optimal control inputs through numerical differentiation (which was the case with `scipy`), we are now computing it through automatic differentiation, using interior-point methods [40] to solve the optimization problem. Given a current kernel and X, Y data, we compute the large matrix inverse $(K + \sigma_\epsilon^2 I)^{-1}$ needed for GP evaluation and keep it in memory. Using the symbolic expression of the GP posterior mean and variance, we can estimate $\hat{x}_{t+i}$ from any given $\hat{x}_{t+i-1}$ and u_{t+i-1} during the optimization procedure. Once the next sequence of control inputs (over a given horizon M) has been selected by the optimization procedure, we go back to regular computations (numerical instead of symbolic), compute the next sequence of true and estimated states, plot them, optimize the hyperparameters of the GP thanks to the new measurements, and start again.

The first step in running these experiments is to ensure the GP is converging. The hyperparameters need to be bounded in order to avoid any numerical issues in case they do not converge well. The kernel and priors on its hyperparameters (hyperpriors) also need to be reasonably chosen from previous experiments. There also need to be sufficient amounts of data available. We also standardize the data: the GP does not receive X as input but $\frac{X-\mu}{\sigma}$, where μ and σ are the input's data mean and standard deviation. This also significantly helps with convergence.

Once the GP is converging to an accurate and robust model, we can focus on the optimization part. The Ipopt library [40] is used to find a local optimum of the nonconvex, nonlinear optimization problem (4.11). Although we can only expect the solver to return a local optimum to this complex problem, several tricks can be used to ensure the local optimum is as good as possible. First of all, Ipopt is better equipped for such problems than classical optimization solvers, for example thanks to the Hessian regularization or the line search for finding an acceptable step size that are applied; see [46] for more details. Also, we start by using the direct single shooting method provided in CasADi's example pack to solve the optimization problem: but it is also possible to use other methods such as direct multiple shooting. The difference

between those two methods, for which there exists very clear documentation, is that with multiple shooting, not only the control inputs are discretized and entered as variables of the NLP problem, but also the states. This can be computationally heavier, but yields better results in terms of the local optimum found by the method. Another useful trick commonly used in nonconvex optimization, is to try a few random initializations for the algorithm and then pick the one that yields the best optimum. This helps with obtaining more robust solutions. We implement several possibilities for initializing the optimization in the code: the user can either start at 0, at the precedent control input, at a random point, or at the point that gave the lowest cost from a few random tries. We observe that after these few tricks have been applied, the local optimum found by Ipopt is most of the time much better than 20 random trajectories we try).

The computational bottlenecks in our procedure are solving (4.11) and training the GP, i.e., updating the covariance matrix with the newly observed data and inverting $K + \sigma_\epsilon^2 I$ in order to make predictions. This would greatly benefit from techniques coming from sparse GPs, which aim at computing a small and informative covariance matrix from all the gathered data. Using multioutput-GPs is also a source of slow computations. For d independent output dimensions, if all d GPs are different (same data but different kernels), then we need to go through a list of them and invert d covariance matrices to make predictions, and run the hyperparameter optimization procedure d times. In our implementation, since using different kernels for each output dimension does not seem to greatly improve the final performance, we decide to use one set of hyperparameters for all output dimensions.

This goes outside the scope of this project and we leave this for future work; but it would be interesting to see how much our method can be sped up with such practical improvements.

5.1.3 Metrics

In order to rigorously compare exploration strategies obtained for different parameters, we need a quantitative measurement of the quality of the learned GP model $\hat{f}$ compared to the true dynamics f. To do so, we define a grid Ω of at least 500 points $(x, u) \in \mathcal{X} \times \mathcal{U}$ uniformly distributed over the region of use of the system. Then, for each point (x, u) in the grid, we compute $\hat{x}_{next} = \hat{f}(x, u)$ (mean of the GP) and $x_{next} = f(x, u)$. We then define the RMSE over Ω as

$$\begin{aligned} RMSE &= \sqrt{\frac{1}{nd_x} \sum_{(x,u)\in\Omega} ||x_{next} - \hat{x}_{next}||^2_{L^2(\Omega)}} \\ &= \sqrt{\frac{1}{nd_x} \sum_{(x,u)\in\Omega} \sum_{i=1}^{d_x} (x_{next,i} - \hat{x}_{next}, i)^2}, \end{aligned} \tag{5.9}$$

where n is the number of points in the grid.

Other metrics can also be derived for measuring how well a GP model has been learned. For example, it can be interesting to evaluate the GP as a whole, including its posterior variance. However, most of the time its posterior mean is used, for example for control. The RMSE of the mean as a prediction is also what is most often considered in literature. Therefore, we focus on this metric for quantifying the prediction accuracy of a GP model after an experiment.

We also consider another metric for evaluating the exploration achieved by a given method. Given the evaluation grid, we can measure how much of that evaluation

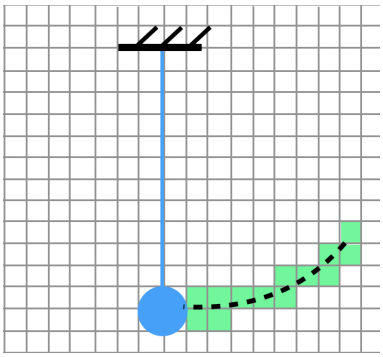


FIGURE 5.4: Illustration of the evaluation grid with schematics of a pendulum. Each output dimension is given a grid, and the number of cells that were actually visited by the system (green) is counted to evaluate the quality of the exploration.

space was explored during a specific simulation by dividing the part of the state space covered by the evaluation grid into small cells, then drawing a histogram of the number of times the true state visited each cell. All positive cells in the histogram are considered visited, and we compute the percentage of the evaluation grid that was visited this way. This is illustrated in Figure 5.4, where the schematics of the pendulum and corresponding evaluation grid are shown. The cells considered visited are marked in green. This gives another, more explicit measure of how well a certain active learning strategy explored the state space, and usually correlates well with the RMSE[1]. We sum up the results of this measure for our benchmark in Table 5.2.

5.1.4 Reducing the variance of the learning results

After running the benchmark experiments 100 times for each baseline with the pendulum, we noticed that even with such a high number of simulations, the variance in the final RMSE metric is quite large. This phenomenon is illustrated in Figure 5.5. By analyzing the results of such experiments in the case of the pendulum, we see that the exploration of the state space is similar for all: a few swing-ups are achieved, a range inside of $[-10; 10]$ of velocities is reached, the control trajectories chosen look similar. However, the observed RMSE itself looks very different, and sometimes even starts to converge before diverging towards high values of error again. This does not appear to reflect a variance in the exploration procedure, but inside the learning procedure: the exploration looks similar each time, but overfitting occurs in the learning part, for example if the hyperparameter optimization of the GP converges to bad parameters. Indeed, good exploration and good learning are correlated but do not necessarily imply one another: our optimal control-based strategies almost always explores more than the random ones, which explains why the mean error in Figure 5.5 is lower, but does not always learn better due to problems with hyperparameter optimization, which explains the high variance.

We aim at reducing this variance in outcomes, in order to better illustrate our main point in the benchmark, which is to show that model-based strategies are superior to the naive ones. Indeed, this variance coming from a lack of robustness in the learning part and not in the exploration part of the framework blurs the difference in exploration between the baselines. Indeed, we were able to test this assumption by

[1] High exploration most of the time means low RMSE, though this is not completely straightforward since the learning procedure can go wrong even when the state space was well explored, and vice versa: the GP estimation of the true dynamics influences the exploration.

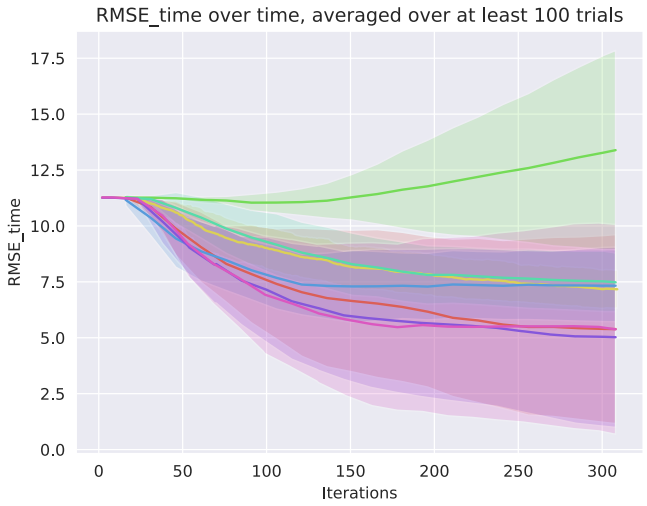


FIGURE 5.5: Box plots of RMSE over time, with the pendulum simulations. We observe a high variance for all methods (no legend, we only want to demonstrate the large variance).

removing the hyperparameter optimization and fixing them to reasonable values; as expected, this showed model-based methods were able to reach lower error.

We try three main solutions in order to robustify the learning procedure, and therefore, avoid overfitting, which causes this extra variance:

- Robustify the hyperparameter optimization itself: by only starting hyperparmeter optimization after a few time steps (we pick 15) and not from the beginning. Thus, we avoid the worst-case scenario of getting stuck in a local minimum early on, which is very likely at the beginning since there is almost no data over which to maximize the marginal log-likelihood. We also use the `optimize_restarts` function from GPy: every time the hyperparameters are optimized, we restart 5 times the optimization procedure from 5 random locations and keep the best result. This also reduces the chances of getting stuck in a local maximum.

- Start with some data or prior knowledge: often in literature, the authors suppose that some prior knowledge is available, either in the form of expert knowledge about the type of system and of kernel to use, or about the hyperparameter values. Such assumptions are common in the GP literature; for example, [11] and related works assumed the value of the hyperparameters is known and fixed. In this work, to provide a fair comparison between all methods, we provide them all with correct but loose hyperpriors. That way, all the data that counts for learning will have been gathered during exploration, hence, the comparison will only focus on how well the methods have explored the input space and how much this has enabled them to learn the estimated dynamics $\hat{f}$; but most of them should have similar, rather correct hyperparameters and none of them should overfit all too often because of such issues not related to exploration.

The problem is then to implement these priors for each dimension of the kernel we use in GPy. We implement a multivariate Gaussian hyperprior in GPy, and make a pull request that is currently still pending. For example, if a successful previous trial with the pendulum (using a squared exponential kernel) ended with variance 20, and lengthscales 0.2, 4, and 150, then we choose a multivariate Gaussian prior with those

values as mean, and 10, 0.1, 1 and 20 as the variance, in order for the Gaussian prior to push towards reasonable values (it is as tightly distributed as the mean is small). We also bound the possible hyperparameter values in a box between 0.001 and 200 to avoid numerical issues, for example during the random optimization restarts.

This method successfully reduces the variance of the learning process. With such hyperparameter priors, the RMSE rarely goes up again after getting some data. The remaining variance in RMSE hence, mostly comes from the exploration itself and not the learning procedure, which enables a fair comparison between the proposed active learning strategies. For example, in the case of the pendulum as seen in Figure 5.6, the distribution of final RMSE with our trajectory optimization method is mostly bimodal: either a swing-up was achieved and the RMSE lies around 2 after 300 data points, or no swing-up was achieved and it lies around 5 or 6. This varying exploration quality explains the variance in this plot, and enables a comparison of how much of the state-space is explored with each method.

Note that using multioutput GPs with one set of hyperparameters per independent output dimension can also improve the performance of the learning procedure, and with good hyperpriors also its robustness. Because of the computational overhead this causes, we only consider one set of hyperparameters, and optimize it to yield the best performance on average over all output dimensions. With our benchmark systems, assuming that the same parameters can fit the data for each output dimension seems reasonable, since removing this assumption did not yield any remarkable increase in performance.

5.2 Benchmark results

The aim of this thesis is to actively learn the dynamics of a controllable dynamical system. Results for all methods listed in Table 4.1 on all systems presented in Section 5.1.1 are shown here.

In order to compare the different methods, we plot mean and standard deviation of RMSE and computation time, over at least 100 trials per method (10 for the opt1 method, which takes a very long time to compute). We attempt to create a benchmark that is as fair as possible, by starting all methods with the same hyperpriors, giving each the same number of data points and horizons, etc. We also present a table summarizing the final results in Table 5.2, in terms of RMSE and percentage of the evaluation space explored. We pick the systems in this benchmark with an eye on what is often used in reinforcement learning, since this paradigm is related to our project, hence, mostly robots and other rigid-body dynamical systems. When choosing the physical parameters of each simulation, we try to pick a realistic system, and to find a problem which is neither too easy to solve (all methods will be able to explore efficiently) nor too hard (none of the presented methods will yield good results). Something in between will be the most interesting for comparing and differentiating between all presented methods.

To obtain these box plots, we linearly interpolate between the results of all 100 runs for each method, and compute mean and standard deviation. Note that all the optimal control methods can also be used with the added control cost $\alpha||u_t||_{L^2}$ as described in 4.2.3, if control costs need to be maintained low or for regularization purposes, or by adding any other costs to the optimization procedure.

TABLE 5.1: List of the methods compared in the benchmark. We give each version an abbreviation to for lighter notations.

Method	Planning horizon	Important parameters	Abbreviation
Standard system identification signals	No planning	Holding time 15 or 10	PRBS, APRBS and chirps
Greedy	$M = 1$	Cost: $H_k(x_k, u_k)$	greedy
Separated search and control	100	$N = 3$ or 4, true dynamics for control	sep1
Separated search and control	100	$N = 3$ or 4, estimated dynamics for control	sep2
Separated search and control	15 or 10	$N = 20$ to 42, true dynamics for control	sep3
Separated search and control	15 or 10	$N = 20$ to 42, estimated dynamics for control	sep4
Optimal control, receding horizon	15 or 10	Cost: $\sum_{i=0}^{M-1} H_k(\hat{x}_{k+i}, u_{k+i})$	opt1
Optimal control, plan and apply	15 or 10	Cost: $\sum_{i=0}^{M-1} H_k(\hat{x}_{k+i}, u_{k+i})$	opt2
Greedy switch	15 or 10	η, cost: $\sum_{i=0}^{M-1} H_k(\hat{x}_{k+i}, u_{k+i})$	switch1
Receding horizon switch	15 or 10	η, cost: $\sum_{i=0}^{M-1} H_k(\hat{x}_{k+i}, u_{k+i})$	switch2

5.2.1 Methods to compare

The methods we compare in this benchmark are presented in Table 5.1. Hereafter, we use the numbers and abbreviations given there to designate those methods.

The experiments are either 300 or 420 time steps long, and $M = 15$ is used for all systems, except the cheetah ($M = 10$). We always show two versions of the separated search and control method, one with about 100 time steps between locations (so $N = 3$ or $N = 4$), and one with the same horizon as the optimal control method for a fair comparison (so $N = 20$ for 300 time steps and $M = 15$, $N = 27$ for 420 time steps and $M = 15$, $N = 30$ for 300 time steps and $M = 10$ and $N = 42$ for 420 time steps and $M = 10$).

5.2.2 Pendulum

The pendulum is the first system on which we test our different strategies, as it is easy to understand and interpret (see illustration in Figure 3.1). As seen in previous sections and among others in Figure 4.3, the quality of the exploration mostly depends on whether a swing-up was achieved. The final RMSE lies around 2 if one was

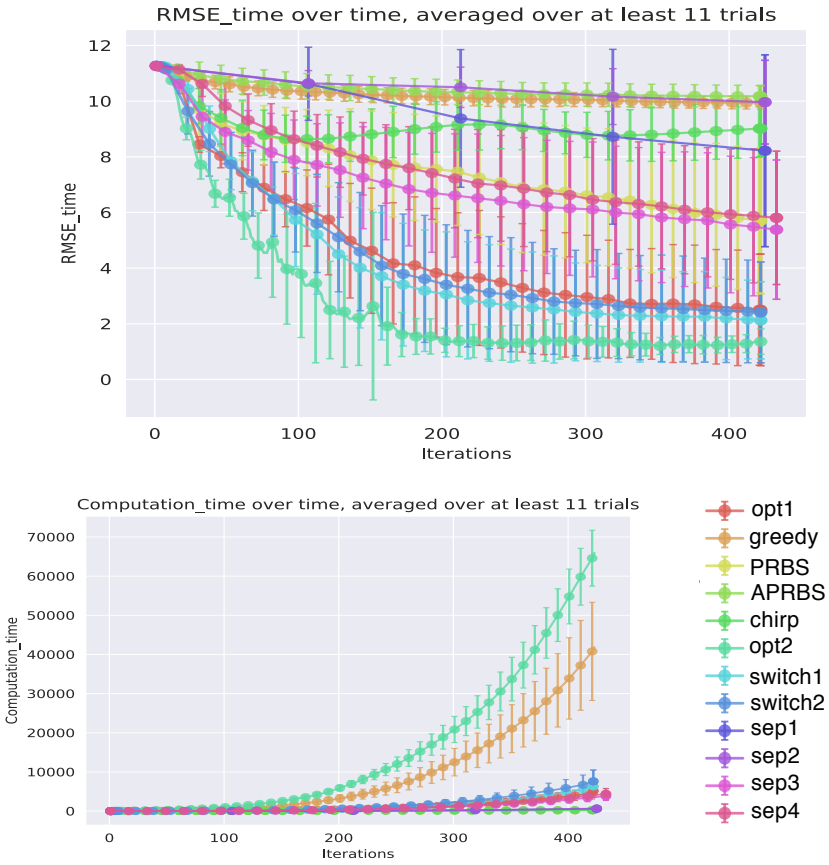


FIGURE 5.6: Box plots of RMSE and computations over time, with the pendulum simulations. We compare all methods listed in 5.2.1. We observe opt1 produces the best results, since a swing-up is almost always achieved, but (opt2) exhibits the best trade-off between computation time and RMSE.

achieved, 4 to 5 otherwise if large oscillations have been observed, and higher if they were smaller. We show the benchmark results for this system in Figure 5.6. In our conditions with $\mathcal{U} = [-3.5; 3.5]$, no swing-up can be achieved with the greedy procedure, instead it necessitates some long-term planning. The classic system identification methods and the greedy one rarely achieve a swing-up, but the optimal control method with receding horizon almost always does. however, it comes at high computational costs; the plan and apply version achieves the best trade-off between computation and performance here. Both switches with $\eta = 0.3$ perform similarly as opt2. The separated search and control method performs reasonably for 20 locations but not as much for only 3; in both cases using the true dynamics for controller yields an increase in performance as expected, but it is not enough to achieve a swing-up. Another set of experiments with shorter running times (300 time steps instead of 420) was also produced, and showed coherent results.

As the pendulum system is easy to learn and very binary in terms of performance (either a swing-up is achieved or not), it is quite easy to differentiate between the active learning strategies presented here. We expect that this difference will be less clear for the other systems in the benchmark since the error results will be more continuous, but that the overall hierarchy of performance will stay consistent.

5.2.3 Two-link planar manipulator

Testing the two-link planar robot (see illustration in Figure 5.1) is a good way to scale up from the pendulum simulations. Here again, the final RMSE mostly depends on whether or not a swing-up was achieved with the first arm. We run these simulations with $\mathcal{U} = [-5; 5]$ and observe results that are a bit more surprising. The optimal control trajectory (opt2) and the switches with $\eta = 0.75$ perform well, but PRBS is able to reach errors that are almost as low. The performance of PRBS can be explained by the fact that this is a rigid-body, toque-controlled system. Hence, most states are linear in the control input; this explains why a method such as PRBS, that explores $\mathcal{X}$ well through high inputs, while not exploring $\mathcal{U}$ (the only values seen for training are $\pm u_{\max}$), can still produce accurate predictions. We also observe (opt2) with a receding horizon is not as efficient as with the pendulum. The separated search and control baseline performs reasonably, but the solution with only few locations ($N = 3$) and known dynamics for control is surprisingly good, better than the same method with many locations ($N = 20$). Both these observations seem to indicate that with this system, control trajectories that go in one direction for a long time are more efficient than fast-reacting strategies. The change in hierarchy of the different strategies shows that the type of system greatly influences the efficiency of one particular active learning strategy; while model-based strategies, such as the separated search and control and the optimal control approaches, are able to consistently produce good results.

Running shorter simulations gave coherent results.

5.2.4 OpenAI double inverted pendulum on a cart

The version of the double inverted pendulum on a cart (DIPC) used in OpenAI Gym (see illustration in Figure 5.3) is very lightly damped, and the automatic resetting starts it with the pendulum balancing on top of the cart. The actuation power is quite large compared to the difficulty of actuating the system, therefore, planning is not necessary for exploring the whole state space. We start by resetting the system at the stable equilibrium (pendulum down) and limiting $\mathcal{U}$ to $[-0.3; 0.3]$, but as shown in Figure 5.8, this is not enough: the system is still very easy to explore, which explains why all the exploration strategies obtain the same result and error at the end.

In order to cope with this problem, we create our own Gym environment based on the DIPC, but add a joint damping parameter of value 8, and use $\mathcal{U} = [-1; 1]$. Now that the system is more damped and harder to actuate, we are able to differentiate between the different methods, as illustrated in Figure 5.9. This time, chirps are the naive solution that perform best, and the separated search and control method does not show very good performance. The greedy solution is also surprisingly good. This can be explained by the nature of the system: with a damped pendulum, it becomes necessary to use the oscillations of the pendulum itself to push it higher, instead of reaching high speeds on the rail and just take the tip of the pendulum up with speed. Hence, pushing the cart in one direction for a long time in order to reach high speeds, like PRBS or the separates search and control methods attempt to do it, does not produce large oscillations. Instead, finding the right frequency and oscillating in one spot with the cart can take the pendulum much higher, though no swing-up is achieved in this case either. This type of behavior is naturally closer to the greedy method and chirp signals, which explains why they perform better than with other systems. The optimal control method and its switched versions with $\eta = 0.13$ adapt to the new system and are also able to leverage its properties by producing fast and

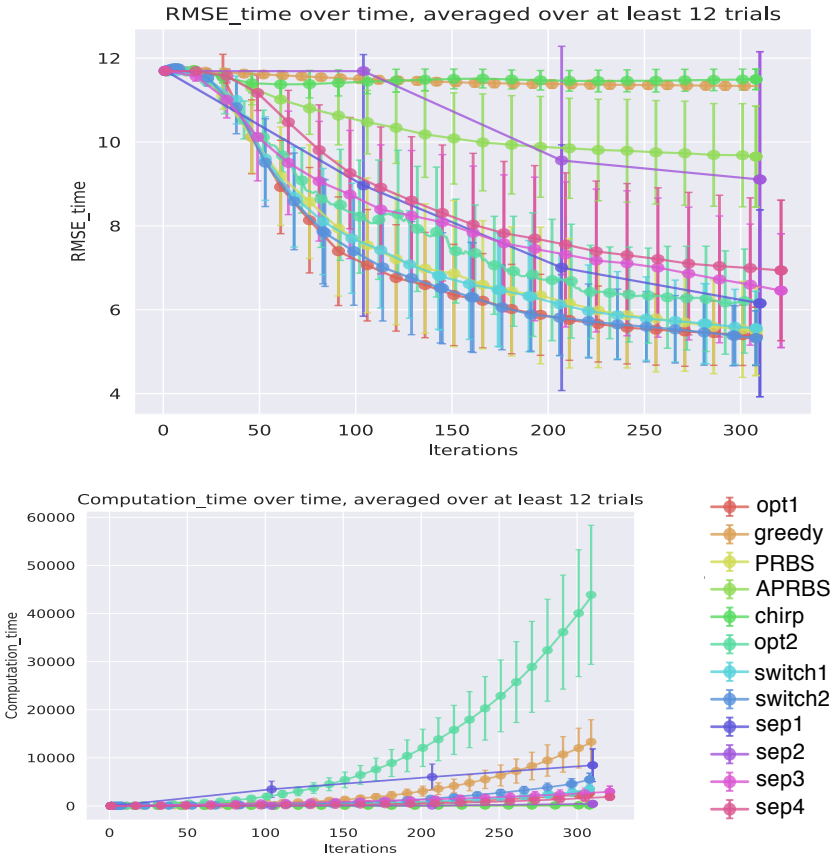


FIGURE 5.7: Box plots of RMSE and computations over time, with simulations of the two-link planar manipulator. We compare all methods listed in 5.2.1. We observe opt1 and its variants produce good results, but PRBS is also able to achieve low error.

large oscillations. Again, the plan and apply version of the optimal control method achieves the best trade-off between performance and computation time, though all of its version (plan and apply, receding horizon, and two switches) end up with the same error. The separated search and control method with few locations does not reach very low error, but it performs better with more locations. Interestingly, for this system the results are slightly better when using estimated dynamics than true dynamics for control.

A second set of simulations with shorter running time was also made, and showed coherent results.

5.2.5 Unicycle

The simulations of the unicycle (see illustration in Figure 5.2) also produce interesting results. This time, it seems the system is hard to actuate for all benchmarked methods with $\mathcal{U} = [-0.05; 0.05]$. The only ones that keep a high error is the separated search and control method with few locations. This method, most of all with estimated dynamics for control, is overfitting a lot, probably because it does not manage to actuate the unicycle much and only makes it turn around itself. This can be explained by the fact that taking a control decision only every 100 steps is too low with this system. Indeed, this type of constrained system can be quite hard to explore, since little actuation power is available, and the system tends to turn around itself instead

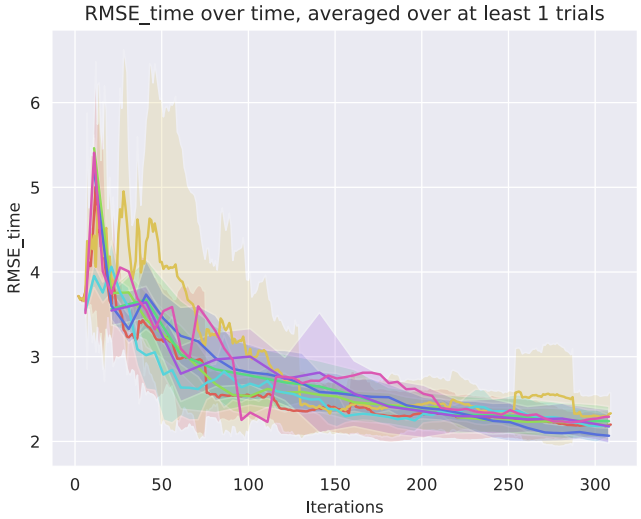


FIGURE 5.8: Box plots of RMSE and computations over time, with the original double inverted pendulum on a cart simulations. The system is very lightly damped and very light, therefore, easy to actuate and explore. Hence, all exploration methods converge to the same solution with approximately the same rate. (No legend, we only want to demonstrate all methods perform similarly.)

of going forward if the control input is not aligned with its forward direction. This is the case in the simulations, where it seems easy to make the unicycle turn but hard for all methods to make it go forward and backwards. As illustrated in Figure 5.10, in the end the optimal control method and its variants still reach the lowest error. A more detailed view, the same plot but without the separated search and control method, is available in Figure 5.10 also.

5.2.6 OpenAI half cheetah

In order to demonstrate the scalability and power of the proposed methods, we also study high dimensional examples in this benchmark. Gym's half cheetah model has $d_x = 18$ states and $d_u = 6$ controls, and requires a certain degree of coordination of the inputs in order to explore the state space (see illustration in Figure 5.3). Indeed, the RMSE drops significantly when the model manages to do a salto and land on its back, is not the case with all methods when using $\mathcal{U} = [-0.55; 0.55]$. Once it is on its back, the RMSE can stagnate or even grow again (overfitting) if it does not move much anymore. The greedy method performs reasonably well with this system, because its characteristic frequency is lower than for the other systems, which is why we use $M = 10$ and not $M = 15$ here. This also reduces the computational overhead of dealing with such a high-dimensional system. The methods based on optimal control perform best again, but this time the plan and apply version is not the most efficient, since waiting 10 time steps between GP updates is probably be too long for such a complex model. On the other hand, both switches are able to reach good performance with reasonable computation costs using $\eta = 1.9$. These results are illustrated in Figure 5.11. With a larger control input, the performance of both switches was able to reach the error achieved by the receding horizon version, as a greedy behavior became more efficient overall. For this system as well as for the pendulum, since the receding horizon version exhibits a lower mean error than the

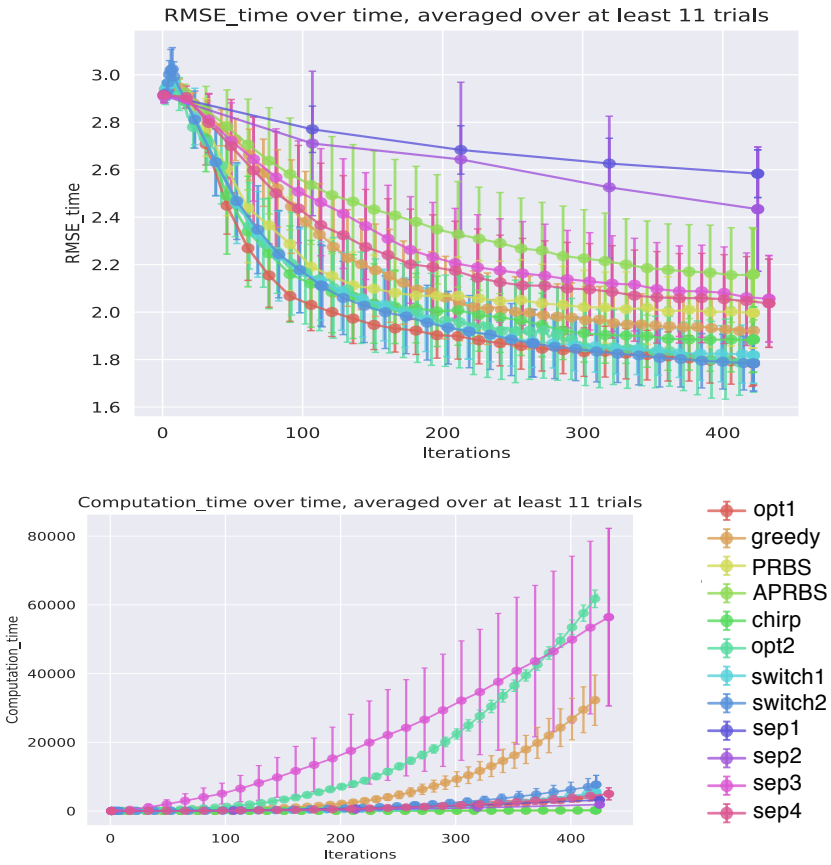


FIGURE 5.9: Box plots of RMSE and computations over time, with our damped DIPC simulations. We compare all methods listed in 5.2.1, and observe the best results for the optimal control method and its variants. Shorter experiments produce coherent results.

plan and apply version, it should be possible to bring both switched methods down to a lower error by lowering η and allowing for more computations, and vice-versa.

Note that it would necessitate more than 400 data points to properly learn such a complex system. Clearly, this requires more sophisticated implementations and serious computational resources, which is out of the scope for this thesis. For example, reinforcement learning methods used with this system typically need a few hundred episodes for training, each episode representing a few hundred of our time steps. And that is just for learning to run with the cheetah, not for learning a full GP model of it. However, the obtained results show the proposed approaches still produce sensitive results. Even if the 400 collected data points are far from enough for learning this system well, the ones produced by the model-based active learning methods are more informative than the ones produced by the naive methods, which is what we aim at demonstrating with this benchmark.

5.2.7 OpenAI ant

The ant model available on Gym is high dimensional and quite complex (see illustration in Figure 5.3), with $d_x = 29$ and $d_u = 8$. This makes it very hard to learn with a GP; we did not manage to find suitable hyperparameters allowing for reasonable prediction error, which is why we do not explicitly include it in this benchmark. Another issue was that we had to add damping to the original Gym model in order

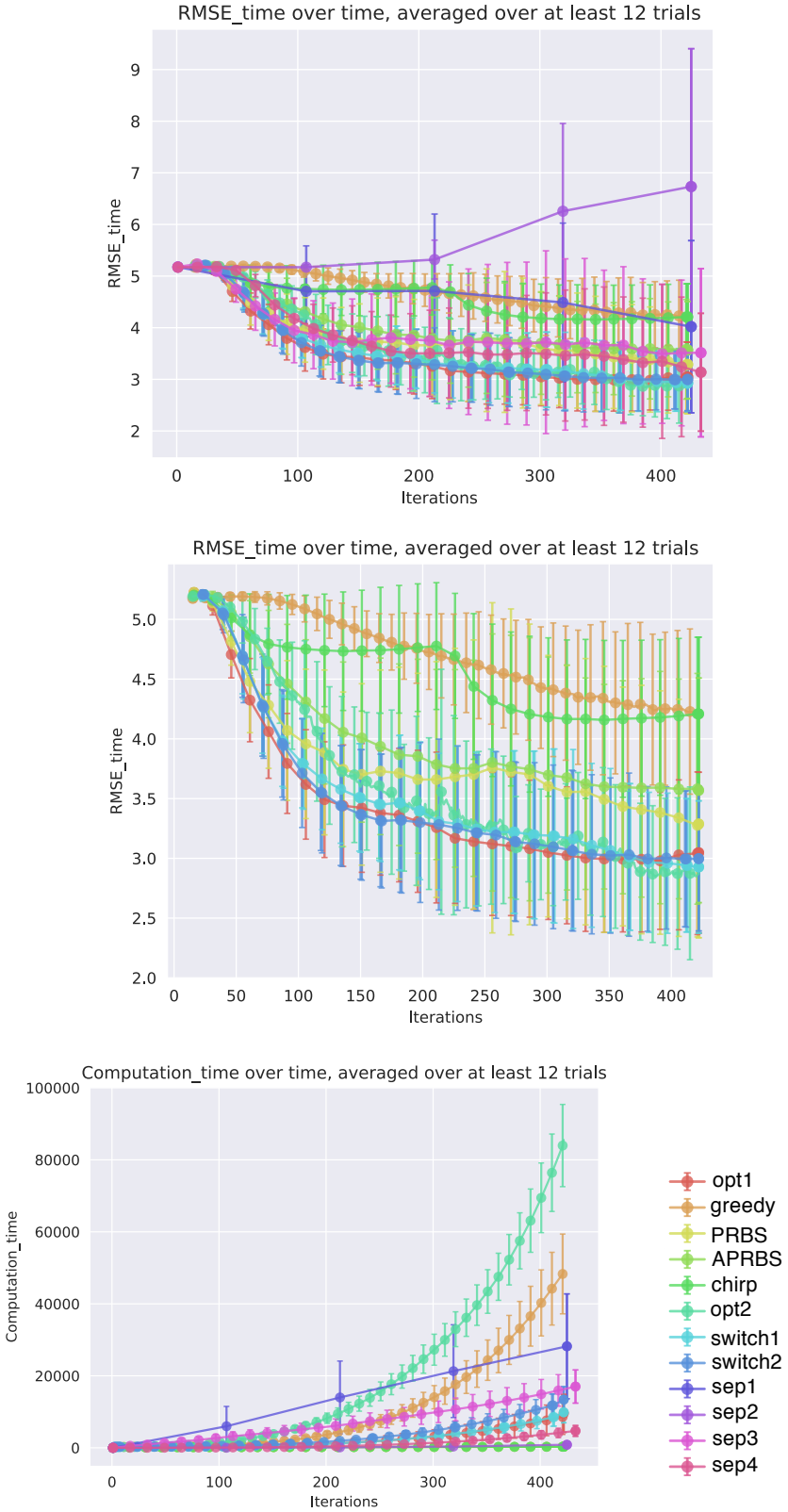


FIGURE 5.10: Box plots of RMSE and computations over time, with the unicycle. We compare all methods listed in 5.2.1 (top), and all except for the separated search and control ones (bottom), for a more detailed view without overfitting.

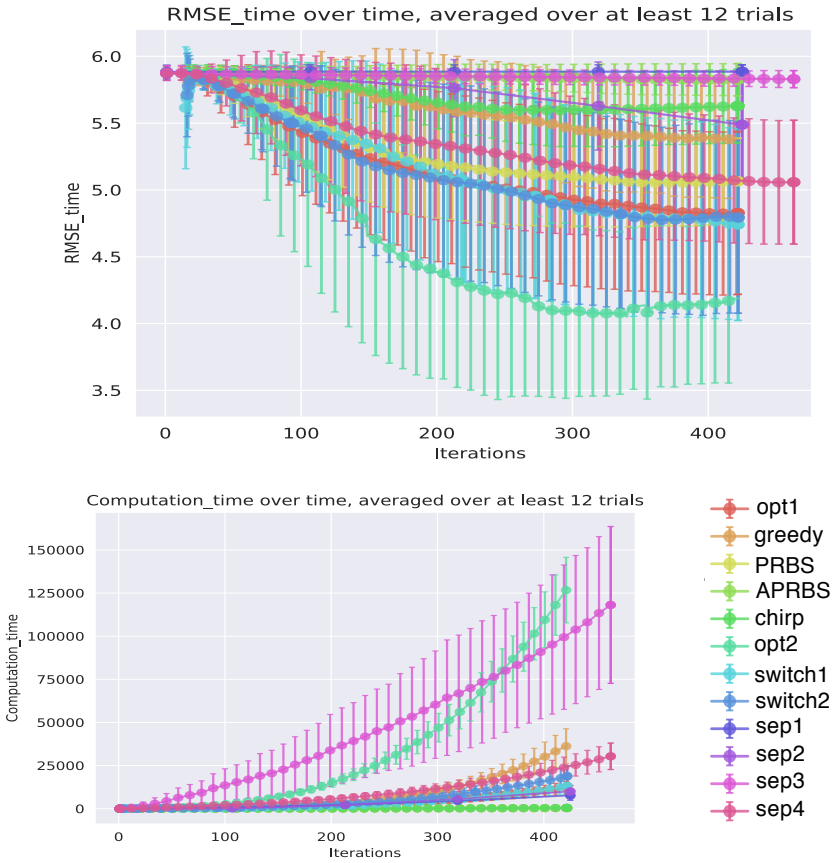


FIGURE 5.11: Box plots of RMSE and computations over time, with half cheetah simulations. We compare all methods listed in 5.2.1, and observe the best results for the optimal control method with receding horizon.

to make it easier to learn for the GP, and then the ant tended to fall on its back, and to not be able to get back up again, which forced us to restart the simulation in that case. However, we did run all proposed methods on this system, hence, showing they can scale up to this system. The exploration results were encouraging, and seemed to indicate that the model-based methods explore the input space more than the naive ones. Similarly to the half cheetah model, learning an accurate GP model of this complex system is a non trivial task itself and is out of scope for this thesis. However, the obtained results show the data gathered by the model-based exploration methods is more informative than naively generated data.

5.2.8 Table comparison of final results

We sum up the final RMSE, and percentage of visited state space for each method on each benchmark system in Table 5.2. These results are coherent with the box plots shown in previous sections, and the overall hierarchy of methods is consistent. This is done for the longer simulations, those with 420 time steps, except for the two-link robot, for which the simulations with 420 time steps are not finished yet, hence, the results are for 300 time steps. We observe that the lowest error is always achieved by one of the variants of the optimal control-based method. Overall, all its versions reach a low error and a high exploration percentage, which seems coherent with the observed behavior. However, we also observe that high exploration percentage and

low error do not always go together: though they are correlated, sometimes a higher exploration percentage was reached by another method with RMSE almost as low, such as for the two-link robot or the unicycle. This can be explained by the fact that it is not only important to explore $\mathcal{X}$ in order to obtain accurate predictions, but also to explore $\mathcal{U}$ and discover coordinated behaviors, which is less often the case with other methods.

TABLE 5.2: Comparing the results on all benchmark systems for all methods. The systems tested are described above (pendulum, two-link robot, double inverted pendulum on a cart (DIPC), unicycle and ant), and the abbreviations of the different methods are given in Table 5.1. We show the final RMSE (mean and standard deviation), and the percentage of explored space on the evaluation grid. Recall these results are averaged over at least 10 trials for opt1, at least 100 trials for all others.

Method used	System tested				
Final RMSE	Pendulum	Two-link	DIPC	Unicycle	Cheetah
PRBS	5.7 ± 2.6	5.5 ± 1.1	2.00 ± 0.15	3.29 ± 0.95	5.1 ± 0.3
APRBS	10.2 ± 0.4	9.7 ± 1.2	2.16 ± 0.19	3.57 ± 0.94	5.8 ± 0.1
chirps	9.0 ± 1.0	11.5 ± 0.2	2.00 ± 0.19	4.21 ± 0.64	5.6 ± 0.3
greedy	10.1 ± 0.2	11.3 ± 0.1	1.88 ± 0.14	4.22 ± 0.69	5.4 ± 0.4
sep1	8.2 ± 3.4	6.1 ± 2.2	2.58 ± 0.10	4.02 ± 1.67	5.9 ± 0.1
sep2	10.0 ± 1.5	9.1 ± 3.0	2.44 ± 0.26	6.73 ± 3.67	5.5 ± 0.4
sep3	5.4 ± 2.5	6.5 ± 1.5	2.09 ± 0.20	3.51 ± 1.63	5.8 ± 0.1
sep4	5.8 ± 2.4	6.9 ± 1.7	2.06 ± 0.18	3.14 ± 1.14	5.1 ± 0.5
opt1	1.4 ± 0.5	6.2 ± 1.3	1.79 ± 0.12	2.87 ± 0.73	4.2 ± 0.6
opt2	2.5 ± 2.0	5.4 ± 0.7	1.79 ± 0.09	3.05 ± 0.67	4.8 ± 0.6
switch1	2.1 ± 1.4	5.6 ± 0.9	1.81 ± 0.11	2.93 ± 0.55	4.7 ± 0.7
switch2	2.4 ± 1.8	5.3 ± 0.7	1.78 ± 0.11	3.00 ± 0.60	4.8 ± 0.7
Explored %					
PRBS	$22.1 \pm 11.0\%$	$49.2 \pm 8.5\%$	$67.4 \pm 8.2\%$	$65.8 \pm 7.3\%$	$50.7 \pm 4.9\%$
APRBS	$8.7 \pm 1.3\%$	$25.0 \pm 6.5\%$	$59.4 \pm 8.0\%$	$62.3 \pm 6.9\%$	$38.4 \pm 3.9\%$
chirps	$14.0 \pm 2.6\%$	$14.8 \pm 3.9\%$	$72.4 \pm 9.0\%$	$41.9 \pm 13.8\%$	$54.4 \pm 9.0\%$
greedy	$10.5 \pm 1.0\%$	$15.7 \pm 1.2\%$	$68.2 \pm 4.6\%$	$51.9 \pm 5.3\%$	$67.2 \pm 3.2\%$
sep1	$17.5 \pm 14.4\%$	$51.6 \pm 12.8\%$	$46.7 \pm 3.5\%$	$65.2 \pm 8.3\%$	$27.3 \pm 1.6\%$
sep2	$10.2 \pm 5.7\%$	$34.9 \pm 11.2\%$	$49.4 \pm 11.1\%$	$56.0 \pm 8.1\%$	$48.1 \pm 7.8\%$
sep3	$23.8 \pm 9.5\%$	$39.1 \pm 7.1\%$	$63.3 \pm 7.0\%$	$66.6 \pm 7.8\%$	$52.7 \pm 3.4\%$
sep4	$22.0 \pm 9.1\%$	$37.7 \pm 8.0\%$	$64.4 \pm 7.5\%$	$66.0 \pm 7.1\%$	$57.2 \pm 4.3\%$
opt1	$46.4 \pm 6.6\%$	$42.6 \pm 6.7\%$	$74.8 \pm 6.5\%$	$65.6 \pm 5.2\%$	$69.1 \pm 3.9\%$
opt2	$37.1 \pm 9.0\%$	$48.2 \pm 7.9\%$	$75.9 \pm 5.3\%$	$66.1 \pm 5.9\%$	$65.3 \pm 3.8\%$
switch1	$37.8 \pm 8.0\%$	$46.5 \pm 7.7\%$	$74.8 \pm 5.7\%$	$65.9 \pm 6.2\%$	$65.6 \pm 3.9\%$
switch2	$38.4 \pm 9.3\%$	$48.1 \pm 7.4\%$	$75.4 \pm 5.9\%$	$65.6 \pm 6.0\%$	$65.7 \pm 3.8\%$

5.2.9 Conclusion on the benchmark

In the last sections, we presented a benchmark of the active learning methods proposed in this thesis. We compare their performance on five different systems (pendulum, two-link robot, double inverted pendulum on a cart (DIPC), unicycle, half

cheetah). The results are summarized in Table 5.2. Next, we analyze and discuss the results of this benchmark.

The method based on optimal control (see Section 4.2) performs best in all systems. Interestingly, the receding horizon version does not always give the best results, because of the greedy behavior it tends to have if the optimal control is not clear. But if it is, for example with the pendulum or the half cheetah, then it performs very well. It usually also has a low variance, but a very high computational price. The plan and apply version consistently performs well for a reasonable computational burden. If the chosen horizon M enables long-term planning while still updating the GP often enough, then this method can explore efficiently since it takes the model explicitly into account, but still avoid the hurdles of becoming greedy by only updating it every M steps. The greedy and receding horizon switches perform similarly most of the time. For systems for which the receding horizon version is more efficient than the plan and apply version (pendulum, half cheetah), both switches can usually reach lower error than the plain plan and apply method for similar computational burden. Overall, we can say that this method performs best, while also being highly versatile, since it can include other tasks as extra terms in the cost function. Its different versions also allow some extra degrees of freedom, since depending on the system one of them can be more efficient than the others, though the plan and apply version is usually a good guess.

The separated search and control approach (see Section 4.1) can also yield good results, as shown with the unicycle or the two-link robot. Since this method does not optimize over the whole control trajectory, but only over a number N of locations to visit, it performs best with systems that can be explored with slow, long movements in one direction or the other (driving the the system to a sequence of unknown locations far apart), instead of fast vibrations like the cheetah or the DIPC. As expected, the lowest error is reached when the number N of locations to visit is high and the true dynamics are used for control. However, sometimes estimated dynamics for control end up reaching higher performance. Overall, it is not as efficient as the optimal control-based method in general, as discussed in Chapter 4.

The greedy method (see Section 3.2) can give good results and reach low error in systems with fast dynamics such as the DIPC or the half cheetah (most of all with higher actuation power), for which vibrating fast can yield good exploration. Because the greedy behavior causes the generated signal to oscillate fast, it can also yield bad results when such fast oscillations are not able to actuate the system much (control frequency too high compared to the system's characteristic frequency); this is the case with the pendulum or the two-link robot.

The classic signals from system identification (PRBS, APRBS, chirps; see Section 2.3.1) showed varying performance, depending on the system's properties. They can perform well if the frequency is well chosen. They also necessitate little computational power, since the only bottleneck is training the GP. However, their performance is not consistent over all systems, and since they do not take the current model into account, they can perform arbitrarily bad. This behavior is equivalent to an open-loop controller, while the model-based active learning methods are closed-loop since they plan depending on the current model. They also do not enable any controlled behavior, since they cannot take any other considerations into account, such as control costs for example. In our benchmark, PRBS performed surprisingly well on average, particularly with the two-link robot. This can be explained by the fact that it can explore the state space efficiently, since it is always uses high valued controls. However, it does not explore $\mathcal{U}$: the only values ever generated are $\pm u_{\max}$. In our benchmark, PRBS sometimes still yields low RMSE even without seeing many values

of $\mathcal{U}$. We suspect that this is because we are only considering systems with rigid-body dynamics and torque control, for which many states are linear in the control input (all angle states for torque-controlled systems). Hence, for all systems except the DIPC, several states are linear in u, which enables accurate predictions even if only two values of u have been learned. The fact that PRBS performs worse on the only benchmarked system for which this is not the case, DIPC (not torque-controlled), supports this interpretation. On top of this, a PRBS-type of behavior is usually not desirable since it can damage systems; the model-based methods can avoid this issue by including control costs in the optimization problem they solve. Overall, we can say that the standard excitation signals from nonlinear system identification are not versatile tools that are guaranteed to perform well for actively learning nonlinear dynamical systems. Though PRBS performs surprisingly well in our benchmark, we believe this is mostly linked to the types of systems that we consider (rigid-body dynamics with torque control).

Chapter 6

Future investigations

In this chapter, we quickly present three ideas that could not be investigated in this thesis, but that could be interesting for future work on this topic.

6.1 Validation of the learned model on a control task

An interesting way of validating the findings of the benchmark experiments would be to select the average learned model for each method, then compare the obtained control performance. We expect that more accurate models, i.e., with lower RMSE, would yield better control performance. This would be a further metric for differentiating between the proposed method, but from a more practical point of view, since it would compare how well the learned model can be used for a concrete control task.

We could also compare the results on using control with the learned model to learning the control task directly from data, with reinforcement learning. It would be interesting to see on a specific example whether or not this direct approach is more sample-efficient. Note that in any case, reinforcement learning remains less general. If a good model of the system has been learned, then it can be used to perform any control task, while each task needs to be learned (more or less) from scratch with reinforcement learning.

6.2 Taking other objectives into account

It would also be beneficial to study how the active learning methods developed in this work can be adapted to take other aspects into account. Adding other terms in the cost function of the optimization problem enables other considerations than pure information gain, as we show by adding control costs to the objective. This versatility of the proposed methods make them suitable for considering related problems, such as exploration while staying close to a reference strategy, safety-conscious active learning, or Bayesian optimization, which is an interesting opening for future work.

6.3 What makes a system easy or hard to explore?

As detailed in Chapter 5, we observed that some systems are easier to explore than others. We identified several properties that characterize this. Intuitively, the model-based approaches are most efficient at exploring a system if the following characteristics are present:

- Enough damping: a system that is very unstable or has very fast dynamics is hard to learn, hence, the model often does not enable accurate predictions for planning. On the other hand, even naive methods can easily explore the whole

state space by adding enough energy into the system. Therefore model-based methods are not as advantageous in this case;

- No boundary effects, that are hard to capture for GP models. For example, in the case of the DIPC, with a system that is very lightly damped, banging against the end of the rail causes several swing-ups, making a PRBS solution near-optimal since banging in one direction produces swing-ups, which the GP cannot identify;

- A sampling rate that is reasonable compared to the characteristic frequency of the system. If the available data is sampled at a rate that is too low compared to the dynamics of the system, then they do not seem continuous anymore and do not reflect the physics of the system. But if they are sampled too fast, then computations soon become very heavy even for just observing a few milliseconds of the physical system;

- Clear milestones that significantly increase prediction accuracy if they are reached. This is the case of the swing-up for the pendulum, or the salto for the cheetah: reaching a certain region of the state space that has very different dynamics reduces the RMSE significantly. This also means that the direction the control should take for active learning is quite clear after some local exploration, hence the obtained behavior is less greedy and more efficient;

- High input dimension: the advantage of the model-based exploration strategies over the naive (random) ones grows with d_u the control dimension. Indeed, in high dimensions visiting interesting regions of the state space requires coordinating the different control inputs, which are all independent in the naive strategies.

The next question to ask is then naturally: can we characterize theoretically what makes a system hard or easy to explore?

Exploring a circular manifold A good way to start this theoretical characterization would be to select a simple dynamical system: one for which the solution in the state space is a circle, e.g., a pendulum. It would be very interesting to take on the one hand a very light pendulum and show it is easy to explore the whole state space, i.e., go around the whole circle, and on the other hand a very heavy pendulum that is hard to explore. We could then try to investigate, given pendulum parameters and $u \in [-u_{\max}; u_{\max}]$, bounds for the expected time before a swing-up is managed, with random control inputs or with our method of trajectory optimization. It would also be interesting to define distances on the manifolds that are implicitly described by the solutions of the dynamics equations of the system.

This should be feasible mathematically, but goes out of scope of the primary objectives of this thesis, therefore we prefer to leave it for future work.

Chapter 7

Conclusion

This thesis tackles the problem of actively learning dynamical systems with Gaussian processes. Efficient methods already exist for actively learning static GPs. However, adding dynamics constrains the problem. Instead of being able to sample directly any data points considered informative by the active learning procedure, it is then necessary to drive the system to a certain state in order to sample data there. Hence, the aim of the dynamic active learning strategy is to generate informative control inputs, that excite the system and drive it to unknown regions of the state space.

Several methods for computing such control trajectories are proposed. First, we consider standard signals used for system identification. A greedy method that optimizes an information-theoretical criterion is then designed, but its greedy behavior does not enable efficient exploration of the input space. Therefore, a separate search and control method is considered, which enables long-term planning and leverages the property of submodularity. It consists in computing informative locations to visit, then driving the system there. Theoretical guarantees on the suboptimality of the sequence of locations to visit can be derived. However, these guarantees do not hold in practice, and the data between the locations is not included in the analysis. Hence, a more efficient approach based on optimal control is proposed, which optimizes over the whole control trajectory and generates highly informative inputs. Several versions of this method are developed, by either running it in a receding horizon fashion, or applying the whole trajectory before re-planning. The receding time version often yields better performance, but is also computationally heavier. We develop an event-trigger switch that leverages both the performance and the computation aspect, by switching between a greedy or receding horizon behavior and a plan and apply strategy.

The performance of the proposed methods is evaluated on a numerical benchmark. We compare all strategies on five systems: an inverted pendulum, a two-link robot manipulator, a double inverted pendulum on a cart, a unicycle, and a half cheetah model. These systems are standard, have varying degrees of complexity, and come among others from OpenAI Gym. This benchmark experimentally shows the power and the performance of the strategy based on optimal control, and offers an interesting comparison between model-free and model-based active learning methods.

This thesis is part of a larger research question. As the interest for data analytics and learning grows, the control community also becomes more interested in using statistical inference for learning systems. Hence, the problem of generating informative data for learning dynamical systems, which means exploring their input and state spaces, is also gaining attention. This thesis aims at providing some insights on how to actively learning dynamical systems, and on the importance of exploration for efficient learning.

Bibliography

[1] A. Fabisch, C. Petzoldt, M. Otto, and F. Kirchner, "A survey of behavior learning applications in robotics - State of the art and perspectives", pp. 1–38, 2018.

[2] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: A survey", *Cognitive Processing*, vol. 12, pp. 319–340, 2011.

[3] T. Graepel, "AlphaGo - Mastering the game of go with deep neural networks and tree search", *Nature*, vol. 529, pp. 484–489, 2016.

[4] M. Deisenroth and C. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search", *Proceedings of the International Conference on Machine Learning - ICML 2011*, vol. 91, pp. 465–472, 2011.

[5] M. Green and J. B. Moore, "Persistence of excitation in linear systems", *Systems and Control Letters*, vol. 7, pp. 351–360, 1986.

[6] J. Schoukens and L. Ljung, "Nonlinear system identification: A user-oriented roadmap", *arXiv 1902.00683*, pp. 1–121, 2019.

[7] A. Emery and A. V. Nenarokomov, "Optimal experiment design", *Measurement Science and Technology*, vol. 9, pp. 864–876, 1998.

[8] A. Jain, T. Nghiem, M. Morari, and R. Mangharam, "Learning and control using Gaussian processes", *Proceedings of the ACM/IEEE International Conference on Cyber-Physical Systems - ICCPS 2018*, pp. 140–149, 2018.

[9] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger, "Information-theoretic regret bounds for Gaussian process optimization in the bandit setting", *IEEE Transactions on Information Theory*, vol. 58, pp. 3250–3265, 2012.

[10] M. Hesse, J. Timmermann, E. Hüllermeier, and A. Trächtler, "A reinforcement learning strategy for the swing-up of the double pendulum on a cart", *Proceedings of the International Conference on System-Integrated Intelligence - ICSII 2018*, pp. 15–20, 2018.

[11] A. Krause, A. P. Singh, and C. Guestrin, "Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies", *Journal of Machine Learning Research*, vol. 9, pp. 235–284, 2008.

[12] A. Krause and C. Guestrin, "Nonmyopic active learning of Gaussian processes", *Proceedings of the International Conference on Machine Learning - ICML 2007*, pp. 449–456, 2007.

[13] J. Binney, A. Krause, and G. S. Sukhatme, "Informative path planning for an autonomous underwater vehicle", *Proceedings of the IEEE International Conference on Robotics and Automation - ICRA 2010*, pp. 4791–4796, 2010.

[14] C. Zimmer, M. Meister, and D. Nguyen-Tuong, "Safe active learning for time-series modeling with Gaussian processes", *Proceedings of the Conference on Advances in Neural Information Processing Systems - NeurIPS 2018*, 2018.

[15] C. Rasmussen and C. Williams, *Gaussian processes for machine learning*. MIT Press, 2006.

[16] D. K. Duvenaud, "Automatic model construction with Gaussian processes", PhD thesis, University of Cambridge, 2014.

[17] I. Steinwart and A. Christmann, *Support Vector Machines*. Springer, New York, NY, 2008.

[18] J. Quiñonero-Candela and C. E. Rasmussen, "A unifying view of sparse approximate Gaussian process regression", *Journal of Machine Learning Research*, vol. 6, pp. 1939–1959, 2005.

[19] M. A. Álvarez, D. Luengo, M. K. Titsias, and N. D. Lawrence, "Variational inducing kernels for sparse convolved multiple output Gaussian processes", *arXiv 0912.3268*, pp. 1–22, 2009.

[20] O. Nelles, *Nonlinear system identification*, 1st ed. Springer-Verlag Berlin Heidelberg, 2001.

[21] C. Guestrin, A. Krause, and A. P. Singh, "Near-optimal sensor placements in Gaussian processes", *Proceedings of the International Conference on Machine Learning - ICML 2005*, vol. 1, pp. 265–272, 2005.

[22] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design", *Proceedings of the International Conference on Machine Learning - ICML 2010*, pp. 1015–1022, 2010.

[23] A. Singh, A. Krause, C. Guestrin, W. Kaiser, and M. Batalin, "Efficient planning of informative paths for multiple robots", *Proceedings of the International Joint Conference on Artifical intelligence - IJCAI 2007*, pp. 2204–2211, 2007.

[24] T. H. Summers, F. L. Cortesi, and J. Lygeros, "On submodularity and controllability in complex dynamical networks", *IEEE Transactions on Control of Network Systems*, vol. 3, pp. 91–101, 2016.

[25] I. Yang, S. A. Burden, R. Rajagopal, S. S. Sastry, and C. J. Tomlin, "Approximation algorithms for optimization of combinatorial dynamical systems", *IEEE Transactions on Automatic Control*, vol. 61, pp. 2644–2649, 2016.

[26] T. Koller, F. Berkenkamp, M. Turchetta, and A. Krause, "Learning-based model predictive control for safe exploration", *Proceedings of the IEEE Conference on Decision and Control - CDC 2018*, 2018.

[27] A. Doerr, C. Daniel, D. Nguyen-Tuong, A. Marco, S. Schaal, T. Marc, and S. Trimpe, "Optimizing long-term predictions for model-based policy search", *Proceedings of the Conference on Robot Learning - CoRL 2017*, vol. 78, pp. 227–238, 2017.

[28] A. Doerr, C. Daniel, M. Schiegg, D. Nguyen-Tuong, S. Schaal, M. Toussaint, and S. Trimpe, "Probabilistic recurrent state-space models", *Proceedings of the International Conference on Machine Learning - ICML 2018*, J. Dy and A. Krause, Eds., pp. 1280–1289, 2018.

[29] T. Xu, Q. Liu, L. Zhao, and J. Peng, "Learning to explore with meta-policy gradient", *Proceedings of the International Conference on Machine Learning - ICML 2018*, 2018.

[30] J. Umenberger, M. Ferizbegovic, T. B. Schön, and H. Hjalmarsson, "Robust exploration in linear quadratic reinforcement learning", *arXiv 1906.01584*, 2019.

[31] M. Ferizbegovic, J. Umenberger, H. Hjalmarsson, and T. B. Schön, "Learning robust LQ-controllers using application oriented exploration", *IEEE Control Systems Letters*, vol. 4, pp. 19–24, 2019.

[32] D. F. Griffiths and D. J. Higham, *Numerical methods for Ordinary Differential Equations*. Springer, London, 2010.

[33] D. Q. Mayne, E. C. Kerrigan, E. J. Van Wyk, and P Falugi, "Tube-based robust nonlinear model predictive control", *International Journal of Robust and Nonlinear Control*, vol. 21, pp. 1341–1353, 2011.

[34] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions-I", *Mathematical Programming*, vol. 14, pp. 265–294, 1978.

[35] A. Krause and C. Guestrin, "Near-optimal nonmyopic value of information in graphical models", *Proceedings of the Conference on Uncertainty in Artificial Intelligence - UAI 2005*, pp. 324–331, 2005.

[36] D. Golovin and A. Krause, "Adaptive submodularity: Theory and applications in active learning and stochastic optimization", *Journal of Artificial Intelligence Research*, vol. 42, pp. 427–486, 2011.

[37] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming", *Proceedings of the IEEE International Conference on Robotics and Automation - ICRA 2014*, pp. 1168–1175, 2014.

[38] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems - IRS 2012*, pp. 4906–4913, 2012.

[39] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control", *Mathematical Programming*, 2018.

[40] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming", *Mathematical Programming*, vol. 106, pp. 25–57, 2005.

[41] L. Wang, *Model predictive control system design and implementation using MATLAB*. Springer, London, 2009.

[42] R. Findeisen, F. Allgöwer, and L. T. Biegler, *Assessment and future directions of nonlinear model-predictive control*. Springer, Berlin, Heidelberg, 2007.

[43] B. Belousov, H. Abdulsamad, M. Schultheis, and J. Peters, "Belief space model predictive control for approximately optimal system identification", Tech. Rep., 2019. [Online]. Available: `https://www.ias.informatik.tu-darmstadt.de/uploads/Team/BorisBelousov/rldm19_belousov.pdf`.

[44] S. Sodhani, A. Goyal, T. Deleu, J. Tang Mila, Y. Bengio, and S. Levine, "Learning powerful policies by using consistent dynamics model", *Proceedings of the Workshop on "Structure and Priors in Reinforcement Learning" at the 2019 International Conference on Learning Representations - ICLR 2019*, pp. 1–6, 2019.

[45] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, "Benchmarking model-based reinforcement learning", *arXiv 1907.02057*, pp. 1–25, 2019.

[46] A. Wächter, "Short tutorial: Getting started with ipopt in 90 minutes", *Dagstuhl Seminar, Combinatorial Scientific Computing*, 2009.