Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Gaussian Processes for Data-Driven Modeling and Control in Robotic Applications

Ph.D. candidate
Fabio Amadio

Advisor
Prof. Ruggero Carli

Co-Advisor
Dr. Alberto Dalla Libera

Director & Coordinator
Prof. Andrea Neviani

Ph.D. School in
Information Engineering

Department of
Information Engineering
University of Padova
2021

University of Padova
Department of Information Engineering

UNIVERSITÀ DEGLI STUDI DI PADOVA

Ph.D course in: Information Engineering
Curriculum: Information Science and Technology
Cycle: XXXIV

Gaussian Processes for Data-Driven Modeling and Control in Robotic Applications

Director: Prof. Andrea Neviani
Advisor: Prof. Ruggero Carli
Co-Advisor: Dr. Alberto Dalla Libera

Ph.D. candidate: Fabio Amadio

Year 2021

Abstract

Over the last years, the domain of robotics is no longer restricted to the mere industrial world and is expanding at great speed into more and more aspects of human life. In order to successfully operate in a huge variety of contexts, the next generation of robots must become more and more autonomous and adaptable. In this context, the application of Machine Learning and Reinforcement Learning techniques turns out to be a promising way to deal with challenges that involve a large-scale and pervasive deployment of robotic systems. Data-driven methods have the potential to equip the robots with the instruments to cope with the uncertainty that characterizes the unstructured environments where they will be increasingly employed. With this regard, Gaussian Processes established themselves as a very powerful and flexible Machine Learning tool. They can be used to solve complex regression problems and directly estimate predictions' uncertainty.

In this thesis, we present Gaussian process-based solutions to different modeling and control problems. In particular, we developed a novel Model-Based Reinforcement Learning algorithm, called *Monte Carlo Probabilistic Inference for Learning COntrol* (MC-PILCO), that can autonomously learn how to perform a certain task, focusing on applications to mechanical systems. In addition, we proposed an alternative version of MC-PILCO that can handle the presence of partial state measures, a typical condition in real-world applications. Furthermore, we proposed a data-driven torque control strategy for robotic systems that exploits Gaussian Process Regression for estimating the robot inverse dynamics. This model is used to close a feedback linearization control loop. Finally, we explored Gaussian Processes' modeling potential by working on high-dimensional dynamics. We used Gaussian Processes to learn a map that projects high-dimensional observations to a latent space of inferior dimension. Then, Gaussian Processes were also employed to estimate a proper transition function inside the latent space that can take into account the effects of control actions. We called the overall structure *Controlled Gaussian Process Latent Variable Model* (CGPDM), and we used it to model the dynamics of a piece of cloth handled by a robotic system. In each of the studied problems, our solutions have been evaluated empirically using real and simulated data.

Sommario

Negli ultimi anni, il campo d'azione della robotica non si limita più alla sola industria, ma si sta espandendo in sempre più aspetti della vita umana. Per operare con successo in questa grande varietà di contesti, la prossima generazione di robot dovrà diventare sempre più autonoma e pronta ad adattarsi a diversi scenari. In tale contesto, le tecniche di *Machine Learning* e *Reinforcement Learning* possono fornire degli strumenti importanti per affrontare le sfide che coinvolgono una diffusione su larga scala dei sistemi robotici. Questi metodi *data-driven* potranno, potenzialmente, dotare i robot dei mezzi per gestire l'incertezza che caratterizza gli ambienti non strutturati nei quali verranno impiegati. A tale riguardo, i *Gaussian Process* si sono affermati come una tecnica di *Machine Learning* molto potente e flessibile. Essi possono essere utilizzati per risolvere complessi problemi di regressione, fornendo direttamente una stima dell'incertezza associata alle previsioni.

In questa tesi, presentiamo soluzioni basate su *Gaussian Process* per diversi problemi di modellazione e controllo. In particolare, abbiamo sviluppato un nuovo algoritmo *Model-Based Reinforcement Learning*, chiamato *Monte Carlo Probabilistic Inference for Learning COntrol* (MC-PILCO), che può imparare autonomamente come controllare un sistema. L'algoritmo è stato poi modificato in modo tale da essere capace di gestire la presenza di misure parziali dello stato, condizione comune quando si lavora su sistemi meccanici. La tesi prosegue presentando una strategia *data-driven* per il controllo di robot, che utilizza dei *Gaussian Process* per stimare la dinamica inversa, utilizzandola poi all'interno di uno schema di controllo basato su *feedback linearization*. Infine, concludiamo esplorando le possibilità offerte dai *Gaussian Process* per la modellizzazione di dinamiche ad alta dimensionalità. Dei *Gaussian Process* sono stati impiegati sia per imparare una mappa che proietta le osservazioni ad alta dimensione in uno spazio latente di dimensione ridotta, che per stimare una funzione di transizione appropriata all'interno di questo spazio latente. Abbiamo chiamato il modello complessivo *Controlled Gaussian Process Latent Variable Model* (CGPDM), e lo abbiamo utilizzato per modellare la dinamica di un pezzo di tessuto manipolato da un robot. In ciascuno dei problemi studiati, le nostre soluzioni sono state valutate empiricamente usando dati reali e simulati.

Acknowledgments

First, I thank my advisor, Prof. Ruggero Carli, for his supervision and for letting me pursue my research interests in autonomy, always providing precious comments and advice.

I would like to deeply thank Dr. Alberto Dalla Libera for his guidance and patience. His competence and initiative have been inspirational and without his help, I am not sure I would have completed this path. I wish him the best for his life and career.

A thank goes to Dr. Diego Romeres, for the fruitful research collaboration established. My thanks also to Dr. Adrià Colomé, with which I started working together back during my master thesis, and that allowed me to have an experience abroad during the Ph.D., despite the strange and difficult times we are all living right now.

I need to thanks all my teammates and the staff of Ruggers Tarvisium, my rugby team. All the emotions lived on the pitch and the fun times spent together are unforgettable.

I thank my friends. You have always been there for me, in the many happy moments, but also in the, luckily few, sad occasions. You truly bring fun and joy to my life.

A special mention goes to my parents, Andreina e Roberto, and my brother, Stefano, which always supported me, no matter what. I am aware of how difficult it could be, sometimes. They thought me that through hard work and passion one could achieve anything. I owe you everything and I will always look up to you.

The last, most special thank goes to my girlfriend Giulia. Your presence and constant support have been fundamental during the last years. I still cannot realize how lucky I was to find you.

Contents

1 Introduction

Nowadays, robots are increasingly being used in many different scenarios, from manufacturing to healthcare, transportation, agriculture, and construction. While in the past robotic applications were mainly focused on the industrial world, today, the domain of robotics is expanding at great speed into more and more aspects of human life. In the future, it is likely to see robotics becoming a pervasive technology in our society. In order to successfully operate in such a huge variety of contexts and inside unstructured environments, the next generation of robots must become more and more autonomous, adaptable and re-configurable.

The development, and large-scale deployment, of robotic systems able to fulfill those requirements, pose numerous challenges. In the first place, robot operation is no longer restricted to a precisely defined environment, as it happens in industrial contexts, where robots are programmed by human experts that precisely know all the task specifications and the geometry of the surroundings. Now, robots must be ready to cope with the inevitable uncertainty that appears in unstructured environments, where items and obstacles are often only partially known, and they could change their positions at any time, as a result of unpredictable external interventions. Also, tasks might involve the manipulation of non-rigid objects, that are particularly hard to model, or even the interaction with human beings, or other autonomous systems.

Furthermore, another crucial aspect to be considered is that only a consistent reduction

of manufacturing and setup costs could make viable a widespread use of robots. Indeed, traditional industrial robots, in order to satisfy precision and repeatability standards, are the results of a high-quality and expensive manufacturing process. Moreover, additional expenses associated with setup activities need to be considered. Production costs may be reduced by the employment of low-quality components, but this is likely to increase measurement noise, uncertainty on the robot's parameters, and the overall wear of the system.

It is evident that building models for such domains purely based on a theoretical understanding of underlying physical principles is infeasible. In fact, standard approaches based on the knowledge of the parameters of both robot and environment might be not enough robust in such an uncertain context, leading to relevant differences between the expected and real behaviour of the system. Machine Learning can offer alternative data-driven solutions for modeling complex robotic tasks and robustly deal with a high level of uncertainties.

Modeling is only one of the possible aspects of robotics where Machine Learning can have a huge impact. Recently, there has been a strong interest in applying data-driven techniques to control learning. In fact, robots, to be truly autonomous, must possess the ability to independently acquire new capabilities. It is not possible to define precisely in advance the correct behaviour for a robot that must operate in an unstructured environment. Hence, the machine must be able to autonomously collect the needed information from available data to adapt its skills to different contexts or even learn how to perform completely new tasks. Reinforcement Learning could point the way towards this level of autonomy. More precisely, Reinforcement Learning is a computational approach that aims at learning how to correctly perform a given task from repeated interactions with the surrounding world. Usually, no knowledge about the environment is available and all the information needed to carry out the task must be obtained by trial and error.

Reinforcement Learning algorithms have already proved able to autonomously solve different control benchmarks. Nevertheless, in order to obtain such results, they required a large amount of experience. This issue is commonly known as data inefficiency and it constitutes a critical problem that hinders the application of Reinforcement Learning to robotic systems. Indeed, collecting a huge amount of data is very problematic when dealing with a real robot: there is a significant cost in performing numerous trials, not to mention the risks of damaging the surrounding environment or the robot itself because of some hazardous movement performed during training. The data inefficiency problem can be mitigated by the so-called Model-Based Reinforcement Learning approach, which aims

at extracting more valuable information from the available data by estimating a model of the environment's dynamics. It has been proved experimentally that the experience required to obtain good solutions can be considerably reduced by exploiting the model during control learning.

The goal of this thesis is to contribute to the development of data-driven methods able to tackle a part of the previously described problems. We focused our efforts on analyzing how probabilistic approaches based on Gaussian Processes can effectively deal with noise and uncertainties in robotic applications. Indeed, Gaussian Processes are a popular class of Machine Learning models, and they constitute a powerful and flexible framework. They are especially advantageous in this context since they can estimate, in a principled way, the uncertainty associated with predictions. In particular, we evaluate different applications of Gaussian Processes in the fields of model-based Reinforcement Learning, dynamical systems modeling, and data-driven control design.

1.1 Manuscript Overview

The thesis is divided into three main parts, each one covering a different problem for which we developed novel algorithms and solutions, all based on Gaussian Processes.

Part I (that comprises Chapters 2, 3, and 4) describes our approach to Model-Based Reinforcement Learning, with a particular focus on applications to mechanical systems. In particular, it illustrates a novel algorithm called *Monte Carlo Probabilistic Inference for Learning COntrol* (MC-PILCO), developed by us. Essentially, the proposed strategy derives a data-driven dynamical model of the environment through Gaussian Process Regression. Then, the algorithm exploits the trained model to update the control policy, based on a Monte Carlo estimation of its gradient. MC-PILCO proved able to robustly solve different control benchmarks, achieving unprecedented levels of data efficiency, and outperforming some of the previous state-of-the-art algorithms. Additionally, we updated MC-PILCO to deal with some of the problems that commonly arise in real-world applications. In particular, when dealing with real systems, it is often impossible to directly measure the whole state of the system. For instance, consider mechanical systems, where velocities are not directly measured but must be reconstructed from the history of position measurements. Thus, we developed *MC-PILCO for Partially Measurable Systems* (MC-PILCO4PMS), a variation of the original algorithm, specifically designed to take into account the presence of state estimators during its operation. This particular strategy was successfully tested on two different real setups. We would like to underline how these kinds of issues are often not properly dealt with in Model-Based Reinforcement

Learning literature, where the majority of the approaches always assume to have direct access to the full state of the system. Finally, we conclude this part by showing another version of the original algorithm that is called *Derivative-Free MC-PILCO* (DF-MC-PILCO). The objective of this last strategy is to solve Reinforcement Learning problems without using velocity estimates. In fact, the tuning of proper velocity estimators might be a tedious task to perform, in particular for systems affected by high measurement noise. DF-MC-PILCO circumvents this problem by working with models and control policies that depend exclusively on position measurements. Experimental results proved that DF-MC-PILCO can achieve performance similar to the one obtained previously by MC-PILCO4PMS.

Part II (that comprises Chapter 5) presents a novel data-driven method for designing robot torque control. We adapted the well-known feedback linearization control scheme to work with Gaussian Process models of the robot dynamics. In particular, we analyzed how to effectively introduce such structures inside the feedback loop to minimize the undesired effects due to the poor generalization properties of the model. The latter can be mitigated by a careful design of the kernel functions. Alternatively, it is possible to exploit the Gaussian Processes to derive estimates of all the different dynamical components used within feedback linearization. We tested this method on a simulated robotic arm, obtaining promising data-driven trajectory tracking controller.

Finally, in Part III (that includes Chapter 6), we present a probabilistic model, always based on Gaussian Processes, designed for dealing with high-dimensional dynamical systems, proving its validity in the context of robotic cloth manipulation. In literature, Gaussian Processes were already being employed for deriving mapping capable of projecting high-dimensional observations to spaces of smaller dimensions (often referred to as latent spaces). In particular, the so-called *Gaussian Process Dynamical Models* (GPDMs) can handle high-dimensional time series, using Gaussian Processes for learning both the latent mapping and a dynamical relationship inside the latent space. Such a method can derive data-driven models for complex dynamics in high-dimensional state spaces, for instance, the behavior of a piece of cloth manipulated by a robot. We extended the GPDM framework by considering the presence of control inputs inside such models and defining the *Controlled Gaussian Process Dynamical Models* (CGPDMs). The proposed method can be applied to learn tractable models for high-dimensional dynamics. The presence of control variables is decisive as it allows to make actual predictions and generalize outside of the training data set. We tested CGPDMs on both a simulated and real cloth manipulation scenario to model the oscillatory dynamics of a piece of fabric.

1.2 Gaussian Process Regression

We conclude the introduction describing the concepts behind Gaussian Process Regression (GPR), as this method lays the foundation of all the model learning strategies that will be illustrated in the next chapters.

One of the most relevant problem that arises in Machine Learning is the, so-called, regression problem. It can be broadly defined as inferring the unknown function that maps input vectors $\boldsymbol{x}$ into real-valued targets y, from a series of observed variables. More precisely, we are given a training set $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$ containing N input-target pairs, where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$ are, respectively, the i-th input vector and the relative target value. Formally, the aim of regression analysis is to learn a model of the noisy relationship

$$y = f(\boldsymbol{x}) + \varepsilon, \tag{1.1}$$

where $f(\cdot)$ is an unknown function and ε is a noise term. Then, given a new input vector $\boldsymbol{x}_*$, the objective of regression is to predict the value assumed by $f(\boldsymbol{x}_*)$.

In the following, we introduce GPs can be used to solve such problem. In Section 1.2.1 we define precisely the GP model in the context of regression, detailing in Section 1.2.2 how to use it to make predictions. Finally, the model training procedure is discussed in Section 1.2.3. For more details about GPR, we refer the reader to Chapter 2 of [97].

1.2.1 Gaussian Process Models

The objective of GPR is to derive a probabilistic model of the function $f(\cdot)$ in Eq. (1.1) from data. A flexible way to place a probability distributions over functions is provided by GPs, that are defined as follows.

Definition. *A* Gaussian Process *is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

A GP is completely characterized by its mean function $m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})]$ (usually assumed to be zero, $m(\boldsymbol{x}) = \boldsymbol{0}$) and covariance function $k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}[(f(\boldsymbol{x}) - m(\boldsymbol{x})) \cdot (f(\boldsymbol{x}') - m(\boldsymbol{x}'))]$ (also called kernel function), and it is usually denoted through the following notation,

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')).$$

In the context of regression the random variables represent the values assumed by the unknown function $f(\boldsymbol{x})$ at input location $\boldsymbol{x}$. Given a set of N input vectors $\boldsymbol{x}_i$ (with $i = 1, \ldots, N$), define $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]$ and $\boldsymbol{f} = [f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N)]^T$. Then, assuming a mean function equal to zero, the GP defines a normal multivariate probability distribution

over the vector of function values,

$$\boldsymbol{f} \sim \mathcal{N}(\mathbf{0}, K(X, X)).$$

The matrix $K(X, X) \in \mathbb{R}^{N \times N}$ is usually called kernel matrix and it is defined element-wise through the kernel function $k(\cdot, \cdot)$. In details, the entry of $K(X, X)$ at row i and column j is equal to $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. A function $k(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ is a valid kernel function if it generates valid covariance matrices. In order to respect this constraint, $k(\cdot, \cdot)$ must be a symmetric and positive semi-definite function. Intuitively, the kernel indicates the degree of similarity between two function values, $f(\boldsymbol{x}_i)$ and $f(\boldsymbol{x}_j)$, depending only on their corresponding inputs. One of the most popular kernel function used in practice is the so-called Squared Exponential (SE) kernel, defined by

$$k_{SE}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \lambda^2 e^{-\frac{1}{2}||\boldsymbol{x}_i - \boldsymbol{x}_j||^2_{\Lambda^{-1}}}.$$

The actual shape of the SE kernel depends on the scaling factor λ and the matrix $\Lambda > 0$, together they are known as the hyperparameters of the kernel function. A standard choice consists in assuming Λ to be diagonal, with diagonal elements called length-scales.

The presence of noise in the relationship of Eq. (1.1) prevent us to directly observe target values $\boldsymbol{f}$. Let y_i be the noisy measurement of target value $f(\boldsymbol{x_i})$, for $i = 1, \ldots, N$, and define $\boldsymbol{y} = [y_1, \ldots, y_N]^T$. If we assume ε to be i.i.d. Gaussian noise, i.e., $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$, the measurement vector $\boldsymbol{y}$ is distributed according to

$$\boldsymbol{y} \sim \mathcal{N}(\mathbf{0}, K(X, X) + \sigma_n^2 I), \tag{1.2}$$

where I is the identity matrix of dimension N. The noise variance σ_n^2 and the kernel's hyperparameters constitute, together, the hyperparameters of the GP model.

1.2.2 Making Predictions

Usually, our primary interest is to incorporate the knowledge that the training data provides about the function and use the GP to make predictions. Let $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$ be the training set composed of observed input vectors and the noisy measurements of the corresponding target values. We are given a test input location $\boldsymbol{x}_*$, our objective is to derive the predictive distribution $p(f(\boldsymbol{x}_*)|\boldsymbol{x}_*, \mathcal{D})$. Since any set of random variables drawn from the GP is jointly Gaussian distributed, from Eq. (1.2), it is possible to write

the following joint prior distribution over noisy observations and test function values,

$$\begin{bmatrix} \boldsymbol{y} \\ f(\boldsymbol{x}_*) \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} K(X,X) + \sigma_n^2 I & \boldsymbol{k}(X, \boldsymbol{x}_*) \\ \boldsymbol{k}(\boldsymbol{x}_*, X) & k(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{bmatrix}\right),$$

with $\boldsymbol{k}(X, \boldsymbol{x}_*) = \boldsymbol{k}(\boldsymbol{x}_*, X)^T = [k(\boldsymbol{x}_1, \boldsymbol{x}_*), \ldots, k(\boldsymbol{x}_N, \boldsymbol{x}_*)]^T$.

The posterior distribution over $f(\boldsymbol{x}_*)$ can be derived by conditioning the joint Gaussian prior on the observations $\boldsymbol{y}$, obtaining the following predictive distribution,

$$f(\boldsymbol{x}_*)|\boldsymbol{x}_*, \mathcal{D} \sim \mathcal{N}(\mu_*, \sigma_*^2), \tag{1.3}$$

with

$$\begin{aligned} \mu_* &= \boldsymbol{k}(\boldsymbol{x}_*, X)(K(X,X) + \sigma_n^2 I)^{-1}\boldsymbol{y}, \\ \sigma_*^2 &= k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}(\boldsymbol{x}_*, X)(K(X,X) + \sigma_n^2 I)^{-1}\boldsymbol{k}(X, \boldsymbol{x}_*). \end{aligned}$$

For more details on the derivation of (1.3), we refer the reader to Appendix 2 of [97].

1.2.3 Learning the Model from Data

Learning a GP model entails finding the most appropriate kernel for the problem, i.e., the form and the parameterization of the covariance function that best explain the training data (formed by a series of observations $\boldsymbol{y}$ and the corresponding input locations X). Concretely, given a parametric covariance function, the objective is to find the set of hyper-parameters that maximizes the marginal likelihood $p(\boldsymbol{y}|X)$. From Eq. (1.2), we can derive its logarithmic expression as,

$$\log(\boldsymbol{y}|X) = -\frac{1}{2}\boldsymbol{y}^T(K(X,X) + \sigma_n^2 I)^{-1}\boldsymbol{y} - \frac{1}{2}\log|K(X,X) + \sigma_n^2 I| - \frac{N}{2}\log(2\pi). \tag{1.4}$$

Then, to train the GP, we maximize expression (1.4) w.r.t. the kernel's hyper-parameters and the measurement noise standard deviation σ_n.

We conclude this discussion by showing in Figure 1.1, as an example, the results of model learning on a GP equipped with SE kernel. The unknown function $f(\cdot)$ we aim to estimate is a sum of sines with random frequencies and amplitudes. Training data are taken on randomly selected input locations applying some Gaussian i.i.d. measurement noise. Note how, on the left part of the plot, the trained GP is able to correctly predict the real function values, with a level of uncertainty that varies depending on the proximity to training points. On the other hand, on the left part of the plot, the mean prediction

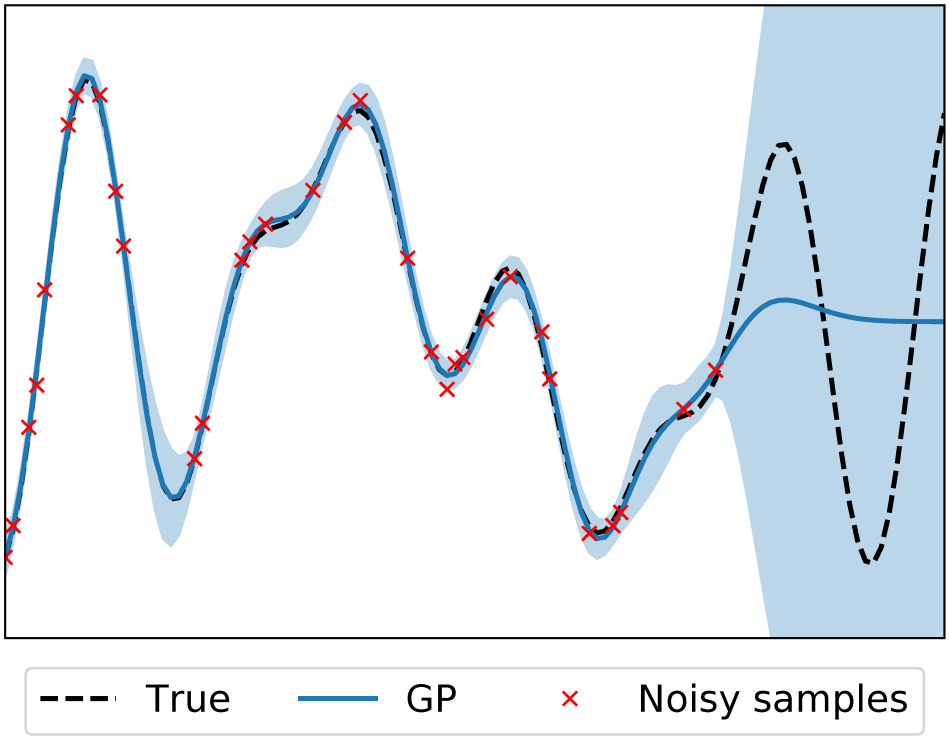


Figure 1.1: GP equipped with SE kernel fitted on some training noisy samples (indicated with red crosses). The dashed line represents the real function we want to infer from data. The mean of the GP prediction is plotted as a solid line, while the shaded area corresponds to an interval of three times the standard deviation.

is 0 (as in the prior) with a huge level of uncertainty that corresponds to the absence of training points in that particular interval of the input space.

Part I

Gaussian Processes for Model-Based Reinforcement Learning

2

Model-Based Policy Search Using Monte Carlo Sampling

In recent years, Reinforcement Learning (RL), [90], has achieved outstanding results in many different environments, and has shown the potential to provide an automated framework for learning different control applications from scratch [1, 48, 58]. However, Model-Free RL (MFRL) algorithms might require a massive amount of interactions with the environment in order to solve the assigned task. This data inefficiency puts a limit to RL's potential in real-world applications, due to the time and cost of interacting with them. In particular, when dealing with mechanical systems, it is critical to learn the task after the least possible amount of trials, to reduce wear and tear and avoid any damage to the system. A promising way to overcome this limit is Model-Based RL (MBRL), which is based on the use of data from interactions to build a predictive model of the environment and exploit it to plan control actions. MBRL increases data efficiency by using the model to extract more valuable information from the available data, [3].

On the other hand, MBRL methods are effective only inasmuch as their models resemble accurately the real systems. Hence, deterministic models might suffer dramatically from model inaccuracy, and the use of stochastic models becomes necessary in order to capture uncertainty. Gaussian Processes (GPs) [97] are a class of Bayesian models commonly used in RL precisely for their intrinsic capability to handle uncertainty

and provide principled stochastic predictions [53][9]. PILCO [33] is a successful MBRL algorithm that uses GP models and gradient-based policy search to achieve substantial data efficiency in solving different control problems, both in simulation as well as with real systems [36][34]. In PILCO, long-term predictions are computed analytically, approximating the distribution of the next state at each time instant with a Gaussian distribution by means of moment matching. In this way, the policy gradient is computed in closed form. However, the use of moment matching introduces also two relevant limitations. (i) Moment matching allows modeling of only unimodal distributions. This fact, besides being a potentially incorrect assumption on the system dynamics, introduces relevant limitations related to initial conditions. In particular, the restriction on the use of unimodal distributions complicates dealing with multimodal initial conditions, as well as being a potential limitation even when the system initial state is unimodal. For instance, in case that the initial variance is high, the optimal solution might be multimodal, due to dependencies on initial conditions. (ii) The computation of the moments is shown to be tractable only when considering Squared Exponential (SE) kernels and differentiable cost functions. In particular, the limitation on the kernel choice might be very stringent, as GPs with SE kernel impose smooth properties on the posterior estimator and might show poor generalization properties in data that have not be seen during training [60, 78, 79, 68].

PILCO has inspired several other MBRL algorithms, which try to improve it in different ways. Limitations due to the use of SE kernels have been addressed in Deep-PILCO [40], where the system evolution is modeled using Bayesian Neural Networks [62] and long-term predictions are computed combining particle-based methods and moment matching. Results show that, compared to PILCO, Deep-PILCO requires a larger number of interactions with the system in order to learn the task. This fact suggests that using neural networks (NNs) might not be advantageous in terms of data-efficiency, due to the considerably high amount of parameters needed to characterize the model. A more articulated approach has been proposed in PETS [20] where the uncertainty of the system dynamics was modeled by means of an ensemble of probabilistic NNs. Despite the positive results in the simulated high-dimension systems, also the numerical results in PETS show that GPs are more data-efficient than NNs when considering low-dimensional systems, such as the cart-pole benchmark. An alternative route has been proposed in [26], where the authors use a simulator to learn a prior for the GP model before starting the RL procedure on the actual system to control. This simulated prior improves the performance of PILCO in areas of the state space with no available data points. However, the method requires an accurate simulator that may not always be available at the user.

Limitations due to the gradient-based optimization were addressed in Black-DROPS [19] which adopts a gradient-free policy optimization. In this way, also non-differentiable cost functions can be used, and the computational time can be improved with the parallelization of the black-box optimizer. With this strategy, Black-DROPS achieves similar data efficiency to PILCO's, but significantly increases asymptotic performance.

Other approaches focused on improving the accuracy of long-term predictions, overcoming approximations due to moment matching. A first attempt has been proposed in [63], where long-term distributions are computed relying on particle-based methods. Based on the current policy and the one-step-ahead GP models, the authors simulate the evolution of a batch of particles sampled from the initial state distribution. Then, the particle trajectories are used to approximate the expected cumulative cost. The policy gradient is computed using the strategy proposed in PEGASUS [67], where by fixing the initial random seed, a probabilistic Markov decision process (MDP) is transformed into an equivalent partially observable MDP with deterministic transitions. Compared to PILCO, results obtained where not satisfactory. The poor performance was attributed to the policy optimization method, in particular, to the inability to escape from the numerous local minima generated by the multimodal distribution. Another particle-based approach is PIPPS [70], where the policy gradient is computed with the so-called *reparameterization trick* [47] instead of the PEGASUS strategy. Given a distribution $p_\theta(\cdot)$, parameterized by θ, the *reparameterization trick* provides an alternative method for generating samples from $p_\theta(\cdot)$ such that those samples are differentiable w.r.t. θ. The *reparameterization trick* has been introduced with successful results in stochastic variational inference (SVI) [47, 76]. In contrast with the results obtained in SVI, where just a few samples are needed to estimate the gradient, the authors of[70] highlighted several issues related to the gradient computed with the *reparameterization trick*, due to its exploding magnitude and random direction. To overcome these issues, they proposed the *total propagation algorithm*, where the *reparameterization trick* is combined with the *likelihood ratio* gradient. The algorithm performs similarly to PILCO with some improvements in the gradient computation, and in the performance in presence of additional noise.

In this chapter, the MBRL algorithm named *Monte Carlo Probabilistic Inference for Learning COntrol* (MC-PILCO) is presented. MC-PILCO is a policy gradient algorithm, which uses GPs to describe the one-step-ahead system dynamics and relies on a particle-based method to approximate the long-term state distribution instead than using moment matching. The gradient of the expected cumulative cost w.r.t. the policy parameters is obtained by *backpropagation* [81] on the associated stochastic computational graph, exploiting the *reparameterization trick*. Differently than in PIPPS, where they focused on

obtaining accurate estimates of the gradient, the optimization problem was interpreted as a stochastic gradient descent (SGD) problem [14]. SGD has been studied in depth in the context of neural networks, where overparameterized models are optimized using noisy estimates of the gradient [57]. Analytical and experimental studies show that the shape of the cost function and the nonlinear activation function adopted can affect dramatically the performance of SGD algorithms [5, 4, 42]. Motivated by the results obtained in this field, w.r.t. the previous particles-based approaches, we considered the use of more complex policies and less peaked cost functions, i.e., less penalizing costs. During policy optimization we also considered the application of *dropout* [89] to the policy parameters, in order to improve the ability to escape from local minima, obtaining more performing policies.

The effectiveness of the proposed choice has been ablated and analyzed in simulation. First, a simulated cart-pole, a common benchmark system, was considered to compare MC-PILCO with PILCO and Black-DROPS. Results show that MC-PILCO outperforms both PILCO and Black-DROPS, which can be considered state-of-the-art GP-based MBRL algorithms. Second, with the purpose to evaluate the behavior of MC-PILCO in a higher-dimensional system, we applied it to a simulated UR5 robotic arm. The considered task consists of learning a joint-space controller able to follow a desired trajectory and it was successfully accomplished. These results confirm that the *reparameterization trick* can be used effectively in MBRL, and Monte Carlo methods do not suffer of gradient estimation problems, as commonly asserted in literature, if properly considering the cost function, the use of *dropout* and richly parameterized policies.

Moroever, differently from previous works which combined GPs with particle-based methods, we show a relevant advantage of this strategy, namely, the possibility of adopting different kernel functions. We considered the use of a kernel function given by the combination of an SE kernel and a polynomial kernel [61], as well as a semiparametrical model [78, 79, 68]. Results show that data efficiency benefits from the use of such kernels, limiting the interaction time required to learn the tasks.

The chapter is structured as follows. In Section 2.1, some background notions are provided: we state the general problem tackled by model-based policy gradient methods, and present modeling approaches of dynamical systems with GPs. In Section 2.2, we present the general characteristics of MC-PILCO, our proposed algorithm, detailing the policy optimization and model learning techniques adopted in Sections 2.3 and 2.4, respectively. In Section 2.5, we analyze several aspects affecting the performance of MC-PILCO, such as the cost shape, *dropout*, and the kernel choice. In Section 2.6, we compare MC-PILCO with PILCO and Black-DROPS using a simulated cart-pole

benchmark system, validate MC-PILCO on a simulated UR5 robot, and also prove the advantages of the particle-based approach when dealing with different distributions of the initial conditions. Finally, conclusions are drawn in Section 2.7.

2.1 Background

In this section, we first introduce the standard framework considered in model-based policy gradient RL methods, and then discuss how to use Gaussian Process Regression (GPR) for model learning. In the latter topic, we focus on three aspects: some background notions about GPR, the description of the model for one-step-ahead predictions, and finally, we discuss long term predictions, focusing on two possible strategies, namely, moment matching and particle-based method.

2.1.1 Model-based policy gradient

Consider the discrete-time system described by the unknown transition function $f(\cdot, \cdot)$,

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{w}_t,$$

where, at each time step t, $\boldsymbol{x}_t \in \mathbb{R}^{d_{\boldsymbol{x}}}$ and $\boldsymbol{u}_t \in \mathbb{R}^{d_{\boldsymbol{u}}}$ are, respectively, the state and the inputs of the system, while $\boldsymbol{w}_t \in \mathbb{R}^{d_{\boldsymbol{x}}}$ is a random variable modeling additive noise.

The cost function $c(\boldsymbol{x}_t)$ is defined to characterize the immediate penalty for being in state $\boldsymbol{x}_t$. Inputs are chosen according to a policy function $\pi_{\boldsymbol{\theta}} : \boldsymbol{x} \mapsto \boldsymbol{u}$ that depends on the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^{d_{\boldsymbol{\theta}}}$. The objective is to find the policy that minimizes the expected cumulative cost over a finite number of time steps T, i.e.,

$$J(\boldsymbol{\theta}) = \sum_{t=0}^{T} \mathbb{E}_{\boldsymbol{x}_t} \left[c(\boldsymbol{x}_t) \right], \tag{2.1}$$

with an initial state distributed according to a given $p(\boldsymbol{x}_0)$. We assume that there is the possibility to always measure the full state of the system, in all its components. The measurements are affected by an unknown additive noise.

A model-based approach for learning a policy consists, generally, of the succession of several trials; i.e. attempts to solve the desired task. Each trial involves the following three main phases:

- *Model Learning*: the data collected from all the previous interactions are used to train a model of the system dynamics (at the first iteration data are collected applying a sequence of exploratory actions, e.g. random controls);

- *Policy Update*: the policy is optimized in order to minimize an estimate of the cost $J(\boldsymbol{\theta})$ obtained by exploiting the trained model;
- *Policy Execution*: the current optimized policy is applied to the system and the data are stored for model improvement.

Model-based policy gradient methods use the learned model to predict how the system state evolves under the control of the current policy $\pi_{\boldsymbol{\theta}}$. These predictions are used to estimate $J(\boldsymbol{\theta})$ and its gradient $\nabla_{\boldsymbol{\theta}} J$, in order to update the policy parameters $\boldsymbol{\theta}$ following a gradient-descent approach.

2.1.2 GPR and one-step-ahead predictions

A common strategy with GPR-based approaches consists of modeling the evolution of each state dimension with a distinct GP that predicts the differences between values at consecutive time steps. As opposed to learning the transition function directly, learning the differences can be advantageous since they are subject to minor variations. Let the state vector be structured as $\boldsymbol{x}_t = \left[x_t^{(1)}, \ldots, x_t^{(d_{\boldsymbol{x}})}\right]^T$. Then, for $i \in \{1, \ldots, d_{\boldsymbol{x}}\}$, $\Delta_t^{(i)} = x_{t+1}^{(i)} - x_t^{(i)}$ denotes the difference between the value of the i-th state component at time $t+1$ and t, and $y_t^{(i)}$ indicates the noisy measurement of $\Delta_t^{(i)}$. Moreover, let $\tilde{\boldsymbol{x}}_t = [\boldsymbol{x}_t^T, \boldsymbol{u}_t^T]^T$ be the vector that includes the state and the input at time t, also called the GP input. Then, given a data set $\mathcal{D} = \left\{\boldsymbol{y}^{(i)}, \tilde{X}\right\}$, where $\boldsymbol{y}^{(i)} = \left[y_{t_1}^{(i)}, \ldots, y_{t_n}^{(i)}\right]^T$ is a vector of n noisy output measurements, and $\tilde{X} = \{\tilde{\boldsymbol{x}}_{t_1}, \ldots, \tilde{\boldsymbol{x}}_{t_n}\}$ is the corresponding set of n GP inputs, GPR assumes the following probabilistic model, for each state dimension $i \in \{1, \ldots, d_x\}$,

$$\boldsymbol{y}^{(i)} = \begin{bmatrix} h^{(i)}(\tilde{\boldsymbol{x}}_{t_1}) \\ \vdots \\ h^{(i)}(\tilde{\boldsymbol{x}}_{t_n}) \end{bmatrix} + \begin{bmatrix} e_{t_1}^{(i)} \\ \vdots \\ e_{t_n}^{(i)} \end{bmatrix} = \boldsymbol{h}^{(i)}(\tilde{X}) + \boldsymbol{e}^{(i)},$$

where $\boldsymbol{e}^{(i)}$ is a zero-mean Gaussian i.i.d. noise with standard deviation σ_i, and $h^{(i)}(\cdot)$ is an unknown function modeled a priori as a zero-mean GP. In particular, we have $\boldsymbol{h}^{(i)} \sim \mathcal{N}(0, K_i(\tilde{X}, \tilde{X}))$, with the a priori covariance matrix $K_i(\tilde{X}, \tilde{X}) \in \mathbb{R}^{n \times n}$ defined element-wise through a kernel function $k_i(\cdot, \cdot)$, namely, the element in j-th row and k-th column is given by $k_i(\tilde{\boldsymbol{x}}_{t_j}, \tilde{\boldsymbol{x}}_{t_k})$. A crucial aspect in GPR is the kernel choice. The kernel function encodes prior assumptions about the process. One of the most common choices for continuous functions is the Squared Exponential (SE) kernel, defined as

$$k_{SE}(\tilde{\boldsymbol{x}}_{t_j}, \tilde{\boldsymbol{x}}_{t_k}) = \lambda^2 e^{-\frac{1}{2}||\tilde{\boldsymbol{x}}_{t_j} - \tilde{\boldsymbol{x}}_{t_k}||^2_{\Lambda^{-1}}}, \tag{2.2}$$

where the scaling factor λ and the matrix Λ are kernel hyper-parameters which can be estimated by marginal likelihood maximization. Typically, Λ is assumed to be diagonal, with the diagonal elements named length-scales. Generally, also the noise standard deviation σ_i is counted among the hyper-parameters of the model.

Remarkably, the posterior distribution of $h^{(i)}(\cdot)$ can be computed in closed form. Let $\tilde{\boldsymbol{x}}_t$ be a general GP input at time t. Then, the distribution of $\Delta_t^{(i)}$, is Gaussian with mean and variance given by

$$\mathbb{E}[\Delta_t^{(i)}] = k_i(\tilde{\boldsymbol{x}}_t, \tilde{X})\Gamma_i^{-1}\boldsymbol{y}^{(i)}, \tag{2.3}$$

$$\mathrm{var}[\Delta_t^{(i)}] = k_i(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_t) - k_i(\tilde{\boldsymbol{x}}_t, \tilde{X})\Gamma_i^{-1}k_i^T(\tilde{\boldsymbol{x}}_t, \tilde{X}), \tag{2.4}$$

with Γ_i and $k_i(\tilde{\boldsymbol{x}}_t, \tilde{X})$ defined as

$$\Gamma_i = (K_i(\tilde{X}, \tilde{X}) + \sigma_i^2 I),$$
$$k_i(\tilde{\boldsymbol{x}}_t, \tilde{X}) = [k_i(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_{t_1}), \ldots, k_i(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_{t_n})].$$

Recalling that the evolution of each state dimension is modeled with a distinct GP, and assuming that the GPs are conditionally independent given the current GP input $\tilde{\boldsymbol{x}}_t$, the posterior distribution for the state at time $t+1$ can be written as

$$p(\boldsymbol{x}_{t+1}|\tilde{\boldsymbol{x}}_t, \mathcal{D}) \sim \mathcal{N}(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}), \tag{2.5}$$

where

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{x}_t + \left[\mathbb{E}[\Delta_t^{(1)}], \ldots, \mathbb{E}[\Delta_t^{(d_{\boldsymbol{x}})}]\right]^T,$$
$$\Sigma_{t+1} = \mathrm{diag}\left(\left[\mathrm{var}[\Delta_t^{(1)}], \ldots, \mathrm{var}[\Delta_t^{(d_{\boldsymbol{x}})}]\right]\right).$$

2.1.3 Long-term predictions with GPs

In MBRL, the policy $\pi_{\boldsymbol{\theta}}$ is evaluated and improved based on long-term predictions of the state evolution: $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$. The exact computation of these quantities entails the application of the one-step-ahead GP models in cascade, considering the propagation of the uncertainty. More precisely, starting from a given initial distribution $p(\boldsymbol{x}_0)$, at each time step t, the next state distribution is obtained by marginalizing (2.5) over $p(\boldsymbol{x}_t)$, i.e,

$$p(\boldsymbol{x}_{t+1}) = \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \mathcal{D})p(\boldsymbol{x}_t)d\boldsymbol{x}_t. \tag{2.6}$$

Unfortunately, computing the exact predicted distribution in (2.6) is not tractable.

There are different ways to solve it approximately, here we discuss two main approaches: moment matching, adopted by PILCO algorithm, and a particle-based method, the strategy followed in this work.

Moment matching. Assuming that the GP models use only the SE kernel as a prior covariance, and considering a normal initial state distribution $p(x_0) \sim \mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0)$, the first and the second moments of $p(\boldsymbol{x}_1)$ can be computed in closed form [35]. Then, the distribution $p(\boldsymbol{x}_1)$ is approximated to be a Gaussian distribution, whose mean and variance correspond to the moments computed previously. Finally, the subsequent probability distributions are computed iterating the procedure for each time step of the prediction horizon. For details about the computation of the first and second moments, we refer the reader to [35]. Moment matching offers the advantage of providing a closed form solution for handling uncertainty propagation through the GP dynamics model. Thus, in this setting, it is possible to analytically compute the policy gradient from long-term predictions. However, the Gaussian approximation performed in moment matching is also the cause of two main weaknesses: (i) The computation of the two moments has been performed assuming the use of SE kernels, which might lead to poor generalization properties in data that have not been seen during training [60, 78, 79, 68]. (ii) Moment matching allows modeling only unimodal distributions, which might be a too restrictive approximation of the real system behavior.

Particle-based method. The integral in (2.6) can be approximated relying on Monte Carlo approaches, in particular on particle-based methods. Specifically, M particles are sampled from the initial state distribution $p(\boldsymbol{x}_0)$. Each one of the M particles is propagated using the one-step-ahead GP models (2.5). Let $\boldsymbol{x}_t^{(m)}$ be the state of the m-th particle at time t, with $m = 1, \ldots, M$. At time step t, the current policy $\pi_{\boldsymbol{\theta}}$ is evaluated to compute the associated control action. The GP model provides the Gaussian distribution $p(\boldsymbol{x}_{t+1}^{(m)} | \boldsymbol{x}_t^{(m)}, \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t^{(m)}), \mathcal{D})$ from which $\boldsymbol{x}_{t+1}^{(m)}$, i.e., the state of the particle at the next time step, is sampled. This process is iterated until a trajectory of length T is generated for each particle. The process is illustrated in Figure 2.1 for the sake of clarity. The long-term distribution at each time step is approximated with the distribution of the particles. Notice that this approach does not impose any constraint on the choice of the kernel function and the initial state distribution. Moreover, no approximations on the distribution $p(\boldsymbol{x}_t)$ are made. Therefore, particle-based methods do not suffer from the problems seen in moment matching.

As regards complexity, the calculation of (2.5) entails several computations of (2.3) and (2.4), which are, respectively, the mean and the variance of the differences between states at consecutive time steps. It can be noted that $\Gamma_i^{-1}\boldsymbol{y}^{(i)}$ is computed a single time

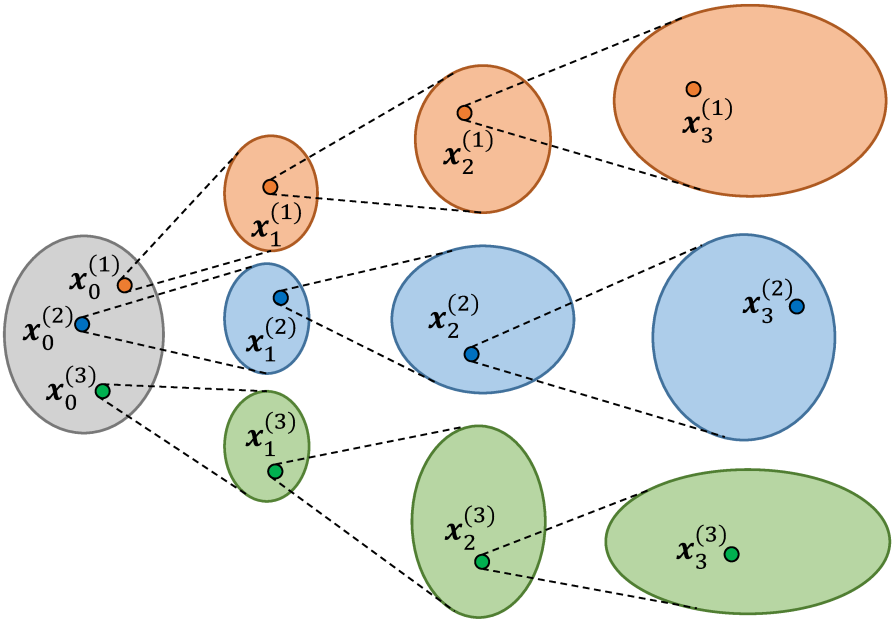


Figure 2.1: Example of three particles propagating through the stochastic model (Gaussian probability distributions are represented as ellipses).

offline during the training of the GP model (same operation is needed in the moment matching case), and the number of operations required to compute (2.3) is linear w.r.t. the number of samples n. The computational bottleneck of particle-based methods is the calculation of (2.4), which is $O(n^2)$. Then, the cost of a single state prediction is $O(d_{\boldsymbol{x}} n^2)$, leading to a total computational cost of $O(d_{\boldsymbol{x}} MTn^2)$. Depending on the complexity of the system dynamics, the number of particles needed to obtain a good approximation might be high, determining a considerable computational burden. Nevertheless, the latter can be substantially mitigated via GPU parallel computing, due to the possibility of computing the evolution of each particle in parallel.

2.2 MC-PILCO

In this section, we present the proposed algorithm. MC-PILCO relies on GPR for model learning and follows a Monte Carlo sampling method to estimate the expected cumulative cost from particles trajectories propagated through the learned model. We exploit the *reparameterization trick* to obtain the policy gradient from the sampled particles and optimize the policy. Compared to the original PILCO algorithm, this strategy is more flexible and avoids the limitations caused by moment matching approximation.

MC-PILCO, in broad terms, consists of the iteration of three main steps, namely,

update the GP models, update the policy parameters, and execute the policy on the system. In its turn, the policy update is composed of three steps, iterated for a maximum of N_{opt} times, namely:

- simulate the evolution of M particles, based on the current $\pi_{\boldsymbol{\theta}}$ and the GPs trained on the previously observed data;
- compute $\hat{J}(\boldsymbol{\theta})$, an approximation of the expected cumulative cost, based on the evolution of the M particles;
- update the policy parameters $\boldsymbol{\theta}$ based on $\nabla_{\boldsymbol{\theta}}\hat{J}(\boldsymbol{\theta})$, the gradient of $\hat{J}(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$, obtained by *backpropagation*.

In the next two sections, we will discuss in more depth the model learning and the policy optimization phases.

2.3 Model Learning

Here, we describe the model learning framework considered in MC-PILCO. We begin by showing the proposed one-step-ahead prediction model and analyzing the advantages w.r.t. the standard model described in Section 2.1.2. Then, we discuss the choice of the kernel functions. Finally, we briefly discuss the model's hyper-parameters optimization and the strategy adopted to reduce the computational burden.

2.3.1 Speed-integration model

Let the state be defined as $\boldsymbol{x}_t = [\boldsymbol{q}_t^T, \dot{\boldsymbol{q}}_t^T]^T$, where $\boldsymbol{q}_t \in \mathbb{R}^{d_{\boldsymbol{q}}}$, with $d_{\boldsymbol{q}} = d_{\boldsymbol{x}}/2$, is the vector of the generalized coordinates of the system at time step t, and, $\dot{\boldsymbol{q}}_t \in \mathbb{R}^{d_{\boldsymbol{q}}}$ represents the derivative of $\boldsymbol{q}_t$ w.r.t. time. MC-PILCO adopts a one-step-ahead model, hereafter denoted as *speed-integration* dynamical model, which exploits the intrinsic correlation between the state components $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. Indeed, when considering a sufficiently small sampling time T_s (small w.r.t. the application), it is reasonable to assume constant accelerations between two consecutive time-steps, obtaining the following evolution of $\boldsymbol{q}_t$,

$$\boldsymbol{q}_{t+1} = \boldsymbol{q}_t + T_s\dot{\boldsymbol{q}}_t + \frac{T_s}{2}(\dot{\boldsymbol{q}}_{t+1} - \dot{\boldsymbol{q}}_t). \tag{2.7}$$

Let $\mathcal{I}_{\boldsymbol{q}}$ (respectively $\mathcal{I}_{\dot{\boldsymbol{q}}}$) be the ordered set of the dimension indices of the state $\boldsymbol{x}$ associated with $\boldsymbol{q}$ (respectively $\dot{\boldsymbol{q}}$). The proposed *speed-integration* model uses the GPR approach described in Section 2.1.2 to train $d_{\boldsymbol{x}}/2$ GPs, each one modeling the evolution

of a distinct velocity component $\Delta_t^{(i_k)}$, with $i_k \in \mathcal{I}_{\dot{\boldsymbol{q}}}$. More specifically, let us rewrite the GP input vector as $\tilde{\boldsymbol{x}}_t = [\boldsymbol{q}_t^T, \dot{\boldsymbol{q}}_t^T, \boldsymbol{u}_t^T]^T$, and indicate with $\dot{\boldsymbol{q}}_{t+1}$ the velocity at the next time step. Then, the *speed-integration* dynamical model samples the predicted velocities from posterior distributions of the form

$$p(\dot{\boldsymbol{q}}_{t+1}|\tilde{\boldsymbol{x}}_t, \mathcal{D}) \sim \mathcal{N}(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}), \tag{2.8}$$

where, for $i_k \in \mathcal{I}_{\dot{\boldsymbol{q}}}$,

$$\begin{aligned}\boldsymbol{\mu}_{t+1} &= \dot{\boldsymbol{q}}_t + \left[\dots, \mathbb{E}[\Delta_t^{(i_k)}], \dots\right]^T, \\ \Sigma_{t+1} &= \text{diag}\left(\left[\dots, \text{var}[\Delta_t^{(i_k)}], \dots\right]\right).\end{aligned}$$

Finally, the evolution of the remaining state components, namely the positions at time step $t+1$, are computed by integrating the sampled velocities, according to (2.7).

Many previous MBRL algorithms, see for instance [33, 19], adopted the standard model described in Section 2.1.2, and hereafter denoted as *full-state* dynamical model. The *full-state* model predicts the change of each state component with a distinct and independent GP. Doing so, the evolution of each state dimension is assumed to be conditionally independent given the current GP input, and it is necessary to learn a number of GPs equal to the state dimension $d_{\boldsymbol{x}}$. Then, compared to the *full-state* model, the proposed *speed-integration* model halves the number of GPs to be trained, decreasing the cost of a state prediction to $O(\frac{d_{\boldsymbol{x}}}{2}MTn^2)$, and associates the evolution of positions and velocities, exploiting their natural relationship. Nevertheless, this approach is based on a constant acceleration assumption, and works properly only when considering small enough sampling times. However, MC-PILCO can use also the standard *full-state* approach, which might be more effective when sampling time is too high.

2.3.2 Kernel functions

Regardless of the GP dynamical model structure adopted, one of the advantages of the particle-based policy optimization method is the possibility of choosing the kernel functions without restrictions. Hence, we considered different kernel functions as examples to model the evolution of physical systems. However, the reader can consider a custom kernel function appropriate for his application.

Squared exponential (SE). The SE kernel described in (2.2) represents the standard choice adopted in many different works. We recall its expression in the following,

$$k_{SE}(\tilde{\boldsymbol{x}}_{t_j}, \tilde{\boldsymbol{x}}_{t_k}) = \lambda^2 e^{-||\tilde{\boldsymbol{x}}_{t_j} - \tilde{\boldsymbol{x}}_{t_k}||^2_{\Lambda^{-1}}}.$$

SE + Polynomial (SE+P$^{(d)}$). Recalling that the sum of kernels is still a kernel [97], we considered also a kernel given by the sum of a SE and a polynomial kernel. In particular, we used the Multiplicative Polynomial (MP) kernel, which is a refinement of the standard polynomial kernel, introduced in [61]. The MP kernel of degree d is defined as the product of d linear kernels, namely,

$$k_P^{(d)}(\tilde{\boldsymbol{x}}_{t_j}, \tilde{\boldsymbol{x}}_{t_k}) := \prod_{r=1}^{d} \left(\sigma_{P_r}^2 + \tilde{\boldsymbol{x}}_{t_j}^T \Sigma_{P_r} \tilde{\boldsymbol{x}}_{t_k}\right).$$

where the $\Sigma_{P_r} > 0$ matrices are distinct diagonal matrices. The diagonal elements of the Σ_{P_r}, together with the $\sigma_{P_r}^2$ elements are the kernel hyper-parameters. The resulting kernel is

$$k_{SE+P^{(d)}}(\tilde{\boldsymbol{x}}_{t_j}, \tilde{\boldsymbol{x}}_{t_k}) = k_{SE}(\tilde{\boldsymbol{x}}_{t_j}, \tilde{\boldsymbol{x}}_{t_k}) + k_P^{(d)}(\tilde{\boldsymbol{x}}_{t_j}, \tilde{\boldsymbol{x}}_{t_k}). \tag{2.9}$$

The idea motivating this choice is the following: the MP kernel allows capturing possible modes of the system that are polynomial functions in $\tilde{\boldsymbol{x}}$, which are typical in mechanical systems [60], while the SE kernel models more complex behaviors not captured by the polynomial structure.

Semi-Parametrical (SP). When prior knowledge about the system dynamics is available, for example given by physics first principles, a Physically Inspired (PI) kernel can be derived. The PI kernel is a linear kernel defined on suitable basis functions $\phi(\tilde{\boldsymbol{x}})$, see for instance [78]. More precisely, $\boldsymbol{\phi}(\tilde{\boldsymbol{x}}) \in \mathbb{R}^{d_\phi}$ is a, possibly nonlinear, transformation of the GP input $\tilde{\boldsymbol{x}}$ determined by the physical model. Then we have

$$k_{PI}(\tilde{\boldsymbol{x}}_{t_j}, \tilde{\boldsymbol{x}}_{t_k}) = \boldsymbol{\phi}^T(\tilde{\boldsymbol{x}}_{t_j}) \Sigma_{PI} \boldsymbol{\phi}(\tilde{\boldsymbol{x}}_{t_k}),$$

where Σ_{PI} is a $d_\phi \times d_\phi$ positive-definite matrix, whose elements are the k_{PI} hyper-parameters; to limit the number of hyper-parameters, a standard choice consists in considering Σ_{PI} to be diagonal. To compensate possible inaccuracies of the physical model, it is common to combine k_{PI} with an SE kernel, obtaining so called semi-parametric kernels [68, 78], expressed as

$$k_{SP}(\tilde{\boldsymbol{x}}_{t_j}, \tilde{\boldsymbol{x}}_{t_k}) = k_{PI}(\tilde{\boldsymbol{x}}_{t_j}, \tilde{\boldsymbol{x}}_{t_k}) + k_{SE}(\tilde{\boldsymbol{x}}_{t_j}, \tilde{\boldsymbol{x}}_{t_k}).$$

The rationale behind this kernel is the following: k_{PI} encodes the prior information given by the physics, and k_{SE} compensates for the dynamical components unmodeled in k_{PI}.

2.3.3 Model optimization and reduction techniques

In MC-PILCO, the GP hyper-parameters are optimized by maximizing the marginal likelihood (ML) of the training samples, see [97]. In Section 2.1.3, we saw that the computational cost of a particle prediction scales with the square of the number of samples n, leading to a considerable computational cost for large n. In this context, it is essential to implement a strategy for limiting the computational burden of a prediction. Several solutions have been proposed in the literature, see [73] for an overview. We implemented a procedure inspired by [25], where the authors proposed an online importance sampling strategy. After optimizing the GP hyper-parameters by ML maximization, the samples in $\mathcal{D}$ are downsampled to a subset $\mathcal{D}_r = \left(\tilde{X}_r, \boldsymbol{y}_r^{(i)}\right)$, which is then used to compute the predictions. This procedure first initializes $\mathcal{D}_r$ with the first sample in $\mathcal{D}$, then, it computes iteratively the GP estimates of all the remaining samples in $\mathcal{D}$, using $\mathcal{D}_r$ as training samples. Each sample in $\mathcal{D}$ is either added to $\mathcal{D}_r$ if the uncertainty of the estimate is higher than a threshold $\beta^{(i)}$ or it is discarded. The GP estimator is updated every time a sample is added to $\mathcal{D}_r$. The trade-off between the reduction of the computational complexity and the severity of the approximation introduced is regulated by tuning $\beta^{(i)}$. The higher the $\beta^{(i)}$, the smaller the number of samples in $\mathcal{D}_r$. On the other hand, using values of $\beta^{(i)}$ too high might compromise the accuracy of GP predictions.

2.4 Policy Optimization

Here, we present the policy optimization strategy adopted in MC-PILCO. We start by describing the general-purpose policy structure considered. Later, we show how to exploit *backpropagation* and the *reparameterization trick* to estimate the policy gradient from particle-based long-term predictions in order to update policy parameters. Finally, we explain how we implemented *dropout* in this framework.

2.4.1 Policy structure

In all the experiments presented in this work, we considered an RBF network policy with outputs limited by an hyperbolic tangent function, properly scaled. We call this function *squashed-RBF-network*, and it is defined as

$$\pi_{\boldsymbol{\theta}}(\boldsymbol{x}) = u_{max} \tanh\left(\frac{1}{u_{max}} \sum_{i=1}^{n_b} w_i e^{||\boldsymbol{a}_i - \boldsymbol{x}||^2_{\Sigma_\pi}}\right). \tag{2.10}$$

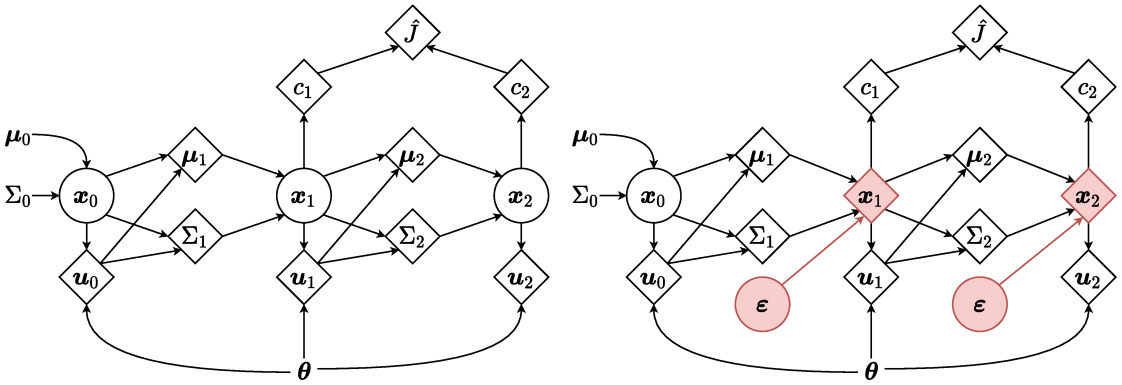
The policy parameters are $\boldsymbol{\theta} = \{\boldsymbol{w}, A, \Sigma_\pi\}$, where $\boldsymbol{w} = [w_1 \dots w_{n_b}]$ and $A = \{\boldsymbol{a}_1 \dots \boldsymbol{a}_{n_b}\}$ are, respectively, the weights and the centers of the Gaussian basis functions, while Σ_π determines the shape of the Gaussian basis functions; in all experiments we assumed Σ_π to be diagonal. The maximum control action u_{max} is constant and chosen depending on the considered system. It is worth mentioning that we are not restricted to this particular choice, and MC-PILCO can easily deal with any kind of differentiable policy.

2.4.2 Policy gradient

MC-PILCO exploits the predictive model illustrated in Section 2.3 to derive an estimate of the expected cumulative cost (2.1), obtained relying on Monte Carlo sampling [17]. Given a control policy $\pi_{\boldsymbol{\theta}}$ and an initial state distribution $p(\boldsymbol{x}_0)$, the evolution of a sufficiently high number of particles is predicted as described in Section 2.1.3. Thus, $\mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)]$ is approximated by the sample mean of the costs incurred by the particles at time step t. Specifically, let $\boldsymbol{x}_t^{(m)}$ be the state of the m-th particle at time t, with $m = 1, \dots, M$ and $t = 0, \dots, T$. The Monte Carlo estimate of the expected cumulative cost is computed with the following expression,

$$\hat{J}(\boldsymbol{\theta}) = \sum_{t=0}^{T} \left(\frac{1}{M} \sum_{m=1}^{M} c\left(\boldsymbol{x}_t^{(m)}\right) \right) . \tag{2.11}$$

To compute the gradient of (2.11) w.r.t. the policy parameters, we use the *reparameterization trick* [47] to differentiate through the stochastic operations. The evolution of every particle $\boldsymbol{x}_t^{(m)}$ at the next time step is sampled from a normal distribution like (2.5), when dealing with *full-state* models, or like (2.8), in case of *speed-integration* models. Hence, the computation of $\hat{J}(\boldsymbol{\theta})$ entails the sampling from probability distributions that depends on policy parameters $\boldsymbol{\theta}$. The presence of such stochastic operations makes the straightforward computation of $\nabla_\theta \hat{J}$ impossible. The *reparameterization trick* allows to still propagate the gradient in such context. In fact, instead of sampling directly from $\mathcal{N}(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1})$, it is possible to sample a point $\boldsymbol{\varepsilon}$ from a zero-mean and unit-variance normal distribution of proper dimensions. Then, $\boldsymbol{\varepsilon}$ can be mapped into the desired distribution as $\boldsymbol{x}_{t+1}^{(m)} = \boldsymbol{\mu}_{t+1} + L_{t+1}\boldsymbol{\varepsilon}$, where L_{t+1} is the Cholesky decomposition of Σ_{t+1}, namely, $L_{t+1}L_{t+1}^T = \Sigma_{t+1}$. In this way, the *reparameterization trick* makes the dependency of $\boldsymbol{x}_{t+1}^{(m)}$ from $\boldsymbol{\theta}$ purely deterministic, allowing to compute $\nabla_{\boldsymbol{\theta}} \hat{J}$ simply by backpropagation. Figure 2.2 illustrates how the *reparameterization trick* works in the context of MC-PILCO. Then, the optimization of the policy parameters $\boldsymbol{\theta}$ can be interpreted as an SGD problem [14], and we adopted the Adam algorithm [46] to solve it (we will denote Adam learning rate with α_{lr}).



(a) Original computational graph. **(b)** Reparameterized computational graph.

Figure 2.2: (Left) Original computational graph of the GP model predictions along two time steps. (Right) Modified computational graph after applying the *reparameterization trick*. Squares and circles represent, respectively, deterministic and stochastic operations.

2.4.3 Dropout of policy weights

To improve exploration in the space of parameters and increase the ability of escaping from local minima during policy optimization, we considered the use of *dropout* [89]. The adopted procedure is described assuming that the policy is the *squashed-RBF-network* in (2.10); similar considerations can be applied to different policy functions. When dropout is applied to the policy in (2.10) at each evaluation of the policy function, the weights $\boldsymbol{w}$ are randomly dropped with probability p_d. This operation is performed by scaling each weight w_i with a random variable $r_i \sim Bernoulli(1-p_d)$, where $Bernoulli(1-p_d)$ denotes a Bernoulli distribution, assuming value $1/(1-p_d)$ with probability $1-p_d$, and 0 with probability p_d. This operation is equivalent to defining a probability distribution for $\boldsymbol{w}$, obtaining a stochastic policy. In particular, as shown in [39], the distribution of each w_i can be approximated with a bimodal distribution, defined by the sum of two properly scaled Gaussians with infinitely small variance ξ^2, namely,

$$p_d \mathcal{N}(0, \xi^2) + (1-p_d)\mathcal{N}\left(\frac{w_i}{1-p_d}, \xi^2\right).$$

The use of a stochastic policy during policy optimization allows increasing the entropy of the particles' distribution. This property increments the probability of visiting low-cost regions and escaping from local minima. In addition, we also verified that dropout can mitigate issues related to exploding gradients. This is probably due to the fact that the average of several values of $\boldsymbol{w}$ is used to compute the gradient, i.e., different policy functions are used, obtaining a regularization of the gradient estimates.

By contrast, the use of a stochastic policy might affect the precision of the obtained

solution due to the additional entropy. We also need to take in consideration that the final objective is to obtain a deterministic policy. For these reasons, we designed an heuristic scaling procedure to gradually decrease the dropout rate, p_d, until it equals 0. The scaling action is triggered by a monitoring signal s, defined from the statistics of the past history of $\hat{J}$. Define the cost change, $\Delta\hat{J}_j = \hat{J}(\boldsymbol{\theta}_j) - \hat{J}(\boldsymbol{\theta}_{j-1})$, where $\boldsymbol{\theta}_j$ denotes the policy parameters at the j-th optimization step. Then, s is computed as a filtered version of the ratio between $\mathcal{E}[\Delta\hat{J}_j]$ and $\sqrt{\mathcal{V}[\Delta\hat{J}_j]}$, that are, respectively, the mean and the standard deviation of $\Delta\hat{J}_j$ computed with an Exponential Moving Average (EMA) filter. The expression of s at the j-th optimization step is the following:

$$s_j = \alpha_s s_{j-1} + (1-\alpha_s)\frac{\mathcal{E}[\Delta\hat{J}_j]}{\sqrt{\mathcal{V}[\Delta\hat{J}_j]}}, \tag{2.12}$$

where

$$\mathcal{E}[\Delta\hat{J}_j] = \alpha_s\mathcal{E}[\Delta\hat{J}_{j-1}] + (1-\alpha_s)\Delta\hat{J}_j,$$
$$\mathcal{V}[\Delta\hat{J}_j] = \alpha_s(\mathcal{V}[\Delta\hat{J}_{j-1}] + (1-\alpha_s)(\Delta\hat{J}_j - \mathcal{E}[\Delta\hat{J}_{j-1}])^2),$$

with α_s a coefficient of the exponential moving average filter, which determines the memory of the filter. At each iteration of the optimization procedure, the algorithm checks if the absolute value of the monitoring signal s in the last n_s iterations is below the threshold σ_s, namely,

$$[|s_{j-n_s}| \dots |s_j|] < \sigma_s, \tag{2.13}$$

where $<$ is an element-wise operator, and the condition in (2.13) is true if it is verified for all the elements. If the condition is verified, p_d is decreased by the quantity Δp_d, and both the learning rate of the optimizer, α_{lr}, and σ_s, are scaled by an arbitrary factor λ_s. Then, we have:

$$p_d = p_d - \Delta p_d \tag{2.14a}$$
$$\alpha_{lr} = \lambda_s \alpha_{lr} \tag{2.14b}$$
$$\sigma_s = \lambda_s \sigma_s \tag{2.14c}$$

The procedure is iterated as long as

$$p_d \geq 0 \text{ and } \alpha_{lr} \geq \alpha_{lr_{min}}, \tag{2.15}$$

where $\alpha_{lr_{min}}$ is the minimum value of the learning rate. When the condition (2.15) is

Parameter	Description	Value
p_d	*dropout probability*	0.25
Δp_d	p_d *reduction coeff.*	0.125
α_{lr}	*Adam learning rate*	0.01
$\alpha_{lr_{min}}$	*minimum learning rate*	0.0025
α_s	*EMA filter coeff.*	0.99
σ_s	*monitoring signal threshold*	0.08
n_s	*num. monitored iterations*	200
λ_s	σ_s *reduction coeff.*	0.5

Table 2.1: Standard values for the policy optimization parameters.

no longer valid, the policy optimization stops and the updated policy can be tested. Table 2.1 reports the optimization parameters used in the majority of the experiments. However, it is worth mentioning that some modifications could be needed in other setups.

The rationale behind this heuristic scaling procedure is the following. The s_j signal is small, if $\mathcal{E}[\Delta\hat{J}_j]$ is close to zero, or if $\mathcal{V}[\Delta\hat{J}_j]$ is particularly high. The first case happens when the optimization reaches a minimum, while the high variance denotes that the particles' trajectories cross regions of the workspace where the uncertainty of the GPs predictions is high. In both cases, we are interested in testing the policy on the real system, in the first case to verify if the configuration reached solves the task, and in the second case to collect data where predictions are uncertain, and so to improve model accuracy. MC-PILCO with dropout is summarized in pseudo-code in Algorithm 1.

Algorithm 1: MC-PILCO

init policy $\pi_{\boldsymbol{\theta}}(\cdot)$, cost $c(\cdot)$, kernel $k(\cdot,\cdot)$, optimization steps N_{opt}, number of particles M, learning rate α_{lr}, min. learning rate $\alpha_{lr_{min}}$, dropout probability p_d, dropout reduction Δ_{p_d} and other monitoring signal parameters: σ_s, λ_s, n_s.
Apply exploratory control actions to system and collect data;
while *task not learned* **do**
  1) Model Learning:
  Train GP models based on collected data;
  2) Policy Update:
  Initialize monitoring signal $s_0 = 0$;
  for $j = 1...N_{opt}$ **do**
    Simulate M particles rollouts with GP models and current policy $\pi_{\boldsymbol{\theta}_j}(\cdot)$;
    Compute $\hat{J}(\boldsymbol{\theta}_j)$ from particles (2.11);
    Compute $\nabla_{\boldsymbol{\theta}}\hat{J}(\boldsymbol{\theta}_j)$ through *backpropagation*;
    $\pi_{\boldsymbol{\theta}_{j+1}}(\cdot) \leftarrow$ Gradient-based policy update (e.g., Adam step);
    Update monitoring signal s_j with (2.12);
    if (2.13) *is True* **then**
      Update p_d, α_{lr} and σ_s with (2.14);
    end
    if (2.15) *is False* **then**
      break;
    end
  end
  3) Policy Execution:
  Apply updated policy to system and collect data;
end
return trained policy, learned GP model;

2.5 Ablation Studies

In this section, we analyze several aspects affecting the performance of MC-PILCO, such as the shape of the cost function, the use of dropout, the kernel choice, and the predictive structure adopted, namely, *full-state* or *speed-integration* dynamical models. The purpose of the analysis is to validate the choices made in the proposed algorithm, and show the effect that they have on the resulting control performance. MC-PILCO has been implemented in Python, exploiting the PyTorch library [71] automatic differentiation functionalities; the code is publicly available [1].

[1] Code available at `https://www.merl.com/research/license/MC-PILCO`

We considered the swing-up of a simulated cart-pole (the system is illustrated in Fig. 2.3), a classical benchmark problem, to perform the ablation studies. The system and the experiments are described in the following. The physical properties of the system are the same considered in the experiments conducted for validating PILCO algorithm [33]: the masses of both cart and pole are 0.5 [kg], the length of the pole is $L = 0.5$ [m], and the coefficient of friction between cart and ground is $b = 0.1$. The state at each time step t is defined as $\boldsymbol{x}_t = [p_t, \dot{p}_t, \theta_t, \dot{\theta}_t]$, where p_t represents the position of the cart and θ_t the angle of the pole. The target states corresponding to the swing-up of the pendulum is given by $p^{des} = 0$ [m] and $|\theta^{des}| = \pi$ [rad]. The downward stable equilibrium point is defined at $\theta_t = 0$ [rad]. As done in [33], in order to avoid singularities due to the angles, $\boldsymbol{x}_t$ is replaced in the algorithm with the state representation $\boldsymbol{x}_t^* = [p_t, \dot{p}_t, \dot{\theta}_t, sin(\theta_t), cos(\theta_t)]$. The control action is the force that pushes the cart horizontally. In all following experiments, we considered white measurement noise with standard deviation of 10^{-2}, and as initial state distribution $\mathcal{N}([0,0,0,0], \text{diag}([10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}]))$. The sampling time is 0.05 seconds. The policy is a *squashed-RBF-network* with $n_b = 200$ basis functions. It receives as input $\boldsymbol{x}_t^*$ and $u_{max} = 10$ [N]. The number of particles has been set to $M = 400$ in all the tests. The exploration trajectory is obtained by applying at each time step t a random control action sampled from $\mathcal{U}(-10, 10)$. GP reduction techniques have not been adopted. The cost function optimized by MC-PILCO is the following,

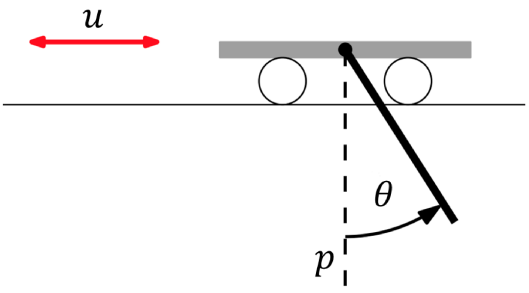


Figure 2.3: Simulated cart-pole system.

$$c(\boldsymbol{x}_t) = 1 - \exp\left(-\left(\frac{|\theta_t| - \pi}{l_\theta}\right)^2 - \left(\frac{p_t}{l_p}\right)^2\right), \tag{2.16}$$

where l_θ and l_p are named length-scales. Notice that the length-scales define the shape of $c(\cdot)$, the cost function goes to its maximum value more rapidly with small length-scales. Therefore, higher cost is associated to the same distance from the target state with lower l_θ and l_p. The lower the length-scale the more selective the cost function. The absolute value on θ_t is needed to allow different swing-up solutions to both the equivalent target angles of the pole π and $-\pi$.

All the comparisons consist in evaluating the results obtained in 50 experiments, each one composed of 5 trials of length 3 seconds. The random seed varies at each experiment, corresponding to different explorations and initialization of the policy, as

well as different realizations of the measurement noise. The performance of the learned policies is evaluated using the cost proposed in [33],

$$c^{pilco}(\boldsymbol{x}_t) = 1 - \exp\left(-\frac{1}{2}\left(\frac{d_t}{0.25}\right)^2\right), \tag{2.17}$$

where $d_t^2 = p_t^2 + 2p_t L sin(\theta_t) + 2L^2(1 + cos(\theta_t))$ is the squared distance between the tip of the pole and its position at the unstable equilibrium point with $p_t = 0$ [m]. We introduce this cost in order to have a common metric to compare both different setups of MC-PILCO and other MBRL algorithms, see Section 2.6.1. For each trial, we report the median values and confidence intervals defined by the 5-th and 95-th percentiles of the cumulative cost obtained with $c^{\text{pilco}}(\cdot)$, as well as the success rates observed. We mark two values of the cumulative cost indicatively associated with an optimal and a sub-optimal swing-up, respectively. A solution is optimal if the pole oscillates only once before reaching the upwards equilibrium. A sub-optimal solution is when the pole oscillates twice. Finally, we label a trial as "success" if $|p_t| < 0.1$ [m] and 170 [deg] $< |\theta_t| <$ 190 [deg] $\forall t$ in the last second of the trial.

2.5.1 Cost shaping

The first test regards the performance obtained varying the length-scales of the cost function in (2.16). Reward shaping is a known important aspect of RL and here we will analyze it for MCPILCO. In Figure 2.4, we compare the evolution of the cumulative costs obtained with $(l_\theta = 3, l_p = 1)$ and $(l_\theta = 0.75, l_p = 0.25)$ and we report the observed success rates. The latter set of length-scales defines a more selective cost as the function shape becomes more skewed. In both cases, we adopted the *speed-integration* model with SE kernel and no dropout was used during policy optimization.

The results show that with $(l_\theta = 3, l_p = 1)$ MC-PILCO performs better. Indeed, with $(l_\theta = 0.75, l_p = 0.25)$ MC-PILCO manages to find a solution only in the 74% of the experiments, while with $(l_\theta = 3, l_p = 1)$ success rate is 82%. Observing the cumulative costs, it is possible to appreciate also a difference in the quality of the policies learned in the two cases. The optimal swing-up can be found in the majority of the experiments only when using $(l_\theta = 3, l_p = 1)$, while it has never been obtained with $(l_\theta = 0.75, l_p = 0.25)$.

This fact suggests that the use of too selective cost functions might decrease significantly the probability of converging to a solution. The reason might be that with small valued length-scales, $c(\boldsymbol{x}_t)$ is very peaked, resulting in almost null gradient, when the policy parameters are far from a good configuration, and increasing the probability of getting stuck in a local minimum. Instead, higher values of the length-scales promote the

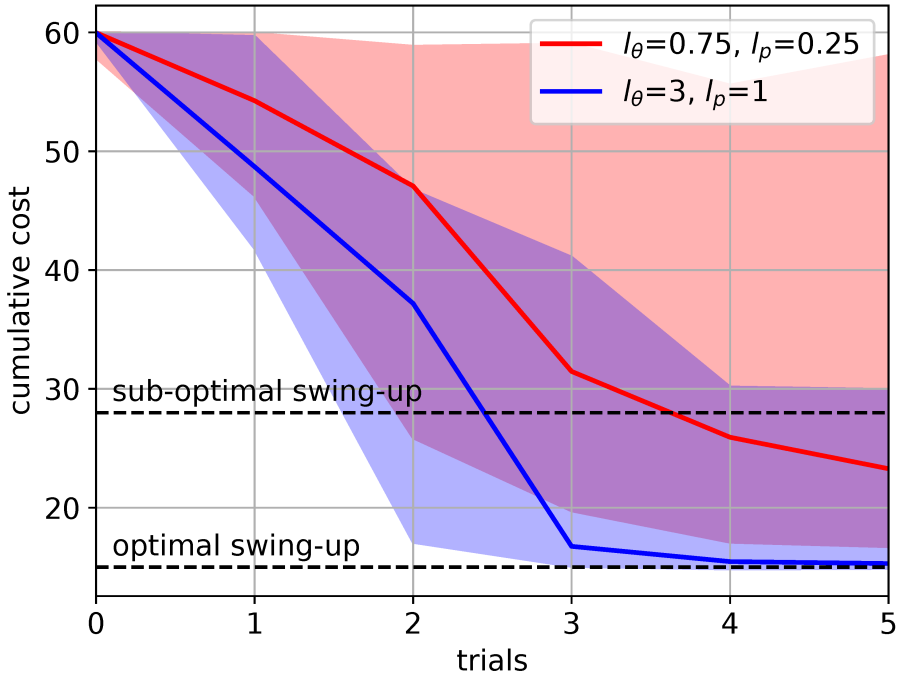


Figure 2.4: Median and confidence interval of the cumulative cost $c^{\text{pilco}}(\cdot)$ per trial obtained using $(l_\theta = 3, l_p = 1)$ or $(l_\theta = 0.75, l_p = 0.25)$. In both cases, we used GP *speed-integration* models with SE kernels and no dropout was applied. Success rates are reported below.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
l=(0.75,0.25)	0%	4%	42%	68%	70%
l=(3,1)	0%	6%	54%	72%	82%

presence of non-null gradients also far away from the objective, facilitating the policy optimization procedure. These observations have already been made in PILCO, but they did not encountered difficulties in using a small length-scale such as 0.25 in (2.17). This may be due to the analytic computation of the policy gradient made possible thanks to moment matching, as well as to the different optimization algorithm used. On the other hand, the length-scales' values seems to have no effect on the precision of the learned solution. To confirm this, in Table 2.2, rows 4-5, are reported the average distances from the target states obtained by successful policies at trial 5 during the last second of interaction. No significant differences appear in term of precision in reaching the targets.

2.5.2 Dropout

In this test, we compared the results obtained using, or not, the dropout during policy optimization. In Figure 2.5, we compare the evolution of the resulting cumulative cost in the two cases and we show the observed success rates.

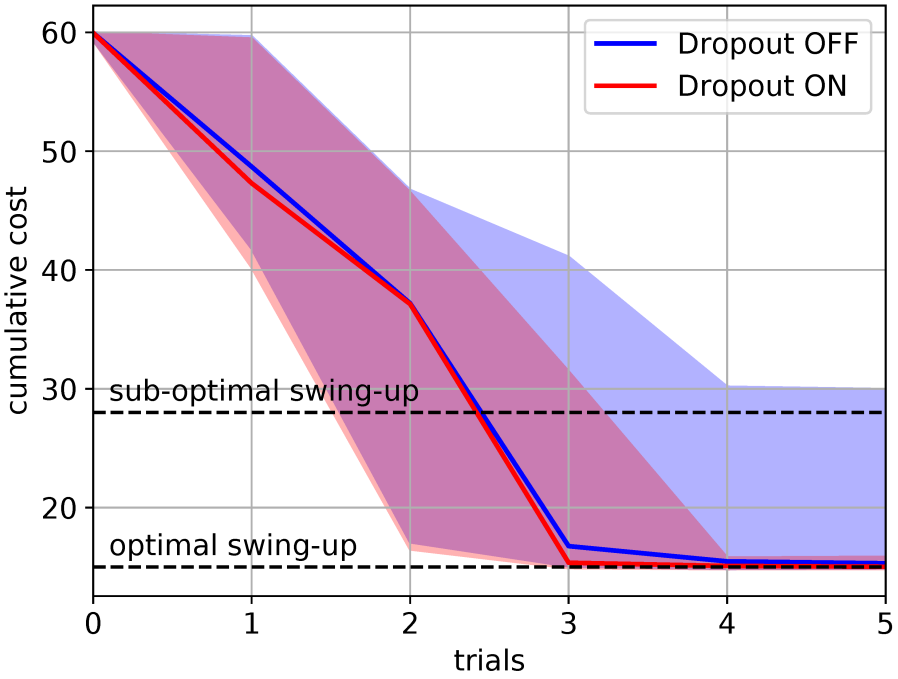


Figure 2.5: Median and confidence interval of the cumulative cost $c^{\text{pilco}}(\cdot)$ per trial obtained using, or not, dropout. In both cases, we adopted GP *speed-integration* model with SE kernels, $l_\theta = 3$ and $l_p = 1$. Success rates are reported below.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
Dropout OFF	0%	6%	54%	72%	82%
Dropout ON	0%	14%	76%	98%	100%

In both scenarios, we adopted the *speed-integration* model with SE kernel and a cost function with length-scales ($l_\theta = 3, l_p = 1$). When using dropout, MC-PILCO learned the optimal solution at trial 4 in the 98% of the experiments, and it managed to obtain it for all the random seeds by trial 5. Instead, without dropout, the optimal policy has not always been found, even in the last trial. Notice that, when dropout is not used, the upper bounds of the cumulative costs in the last two trials are higher, and the task cannot be always solved. Additionally, Table 2.2, rows 3-5, shows that using dropout also helps in decreasing the cart positioning error at the end of the swing-up (in terms of both mean and standard deviation).

Empirically, we found that dropout not only helps in stabilizing the learning process and in finding better solutions more consistently, but it can also improve the precision of the learned policies.

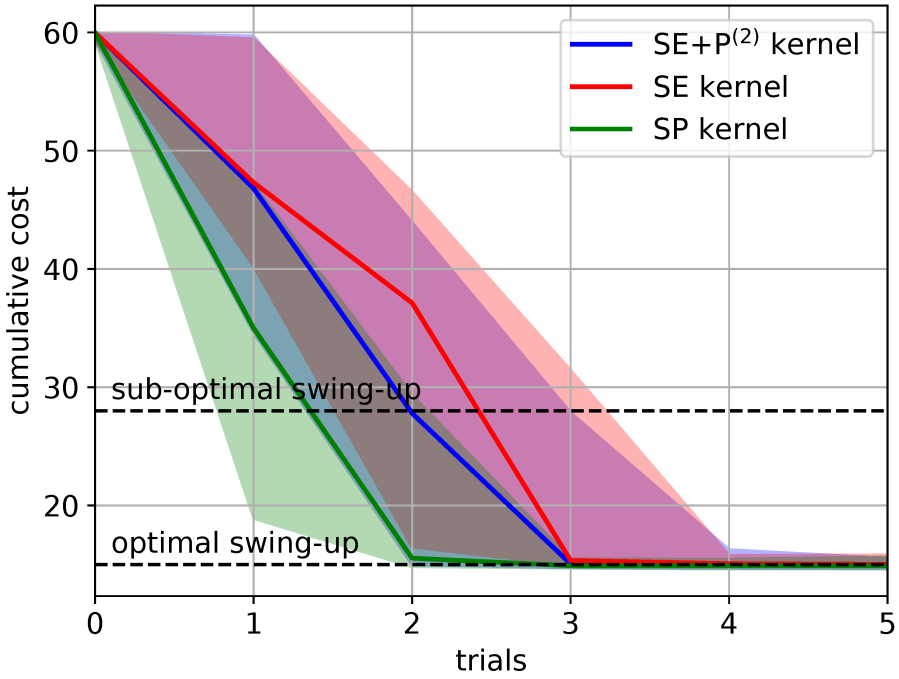


Figure 2.6: Median and confidence interval of the cumulative cost $c^{\text{pilco}}(\cdot)$ per trial obtained using GP *speed-integration* model with kernels SE, SE+P$^{(2)}$ and SP. In all the cases, $l_\theta = 3$, $l_p = 1$, and dropout was used. Success rates are reported below.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
SE	0%	14%	76%	98%	100%
SE+P$^{(2)}$	0%	36%	88%	98%	100%
SP	8%	70%	100%	100%	100%

2.5.3 Kernel function

In this test, we compared the results obtained using the three kernels described in Section 2.3.2, namely the Squared Exponential (SE), the sum of Squared Exponential and Polynomial of second order, (SE+P$^{(2)}$), and the Semi-Parametrical [2] (SP) kernels. In all the cases, we adopted a *speed-integration* model, the cost function was defined with length-scales ($l_\theta = 3, l_p = 1$), and dropout was used.

In Figure 2.6 we can observe that the use of structured kernels, such as SP and SE+P$^{(2)}$, can be beneficial in terms of data efficiency, compared to adopting the standard SE kernel. In fact, the fastest convergence is observed in the SP case, where a success rate of 100% is obtained at trial 3, after only 9 seconds of experience. Also in the

[2]SP basis functions are obtained by isolating, in each ODE defining cart-pole laws of motion, all the state-dependent components that are linearly related. In particular, we have $\phi_{\dot{p}}(\boldsymbol{x}, u) = [\dot{\theta}^2 \ sin(\theta), sin(\theta)cos(\theta), u, \dot{x}]$ for the cart velocity GP, and $\phi_{\dot{\theta}}(\boldsymbol{x}, u) = [\dot{\theta}^2 \ sin(\theta)cos(\theta), sin(\theta), u \ cos(\theta), \dot{x} \ cos(\theta)]$ for the pole velocity GP (details in Appendix A.1).

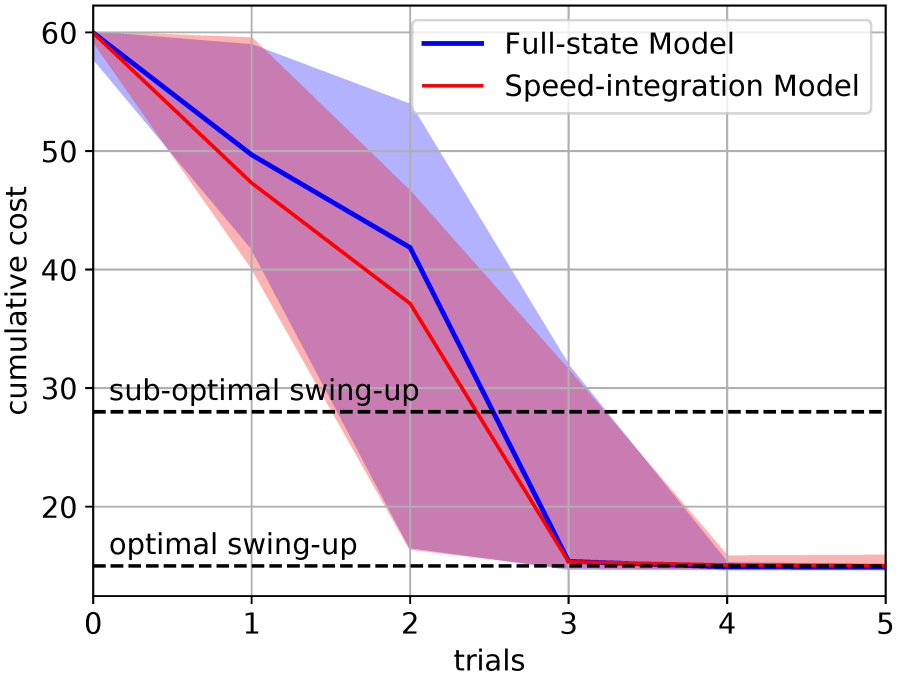


Figure 2.7: Median and confidence interval of the cumulative cost $c^{\text{pilco}}(\cdot)$ per trial obtained using *full-state* or *speed-integration* dynamical model. The kernels are in both cases SE, $l_\theta = 3$, $l_p = 1$ and dropout was used. Success rates are reported below.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
Full-state	0%	12%	70%	98%	100%
Speed-int.	0%	14%	76%	98%	100%

SE+P$^{(2)}$ case, better success rates were obtained at trial 2 and 3, compared to the SE case. This difference can be explained by the capacity of a more structured kernel to better generalize outside of the training set, i.e., to learn a correct enough dynamics of the system also in areas of the state-action space with no available data points.

In fact, some parts of the dynamics of the cart-pole system are polynomial functions of the GP input $\tilde{\boldsymbol{x}}_t = (\boldsymbol{x}_t^*, \boldsymbol{u}_t)$ and the structure of the SE+P$^{(2)}$ achieves better data-efficiency during the first trials, compared to SE. On the other hand, in the case SP kernel, we are exploiting features determined by a direct knowledge of the physical model, thus it is possible to reach a even higher level of data-efficiency.

2.5.4 Speed-integration model

In this test, we compared the performance obtained by the proposed *speed-integration* dynamical model and by the standard *full-state* model. In both cases, SE kernels were chosen, the cost function was defined with length-scales ($l_\theta = 3, l_p = 1$), and dropout

was used. Figure 2.7 shows that the *speed-integration* model obtains better performance at trials 2 and 3, with narrower confidence intervals and better success rates. On the contrary, during the last two trials, the success rate of the *full-state* model slightly improves.

Recall that in the *full-state* model, positions and velocities are learned independently, while, in the *speed-integration* model, the position is computed as the integral of the velocity under a constant acceleration assumption. Then, the *speed-integration* model may reduce uncertainty in long-term predictions and facilitate the learning w.r.t. the counterpart when few data points have been collected. In fact, the *full-state* model could face some difficulties in learning the relationship between positions and respective velocities from a limited amount of data. This reduction of the uncertainty may explain the narrower confidence intervals observed during the first trials of the experiment. On the other hand, when enough data points have been collected (trials 4 and 5), the improvement in precision obtained by *full-state* model is not very significant. Even with comparable performance, the choice of the *speed-integration* model is justified since it halves the number of GPs to learn, hence this structure is also improving the computational time.

2.6 Experiments in Simulation

In this section, two simulated systems are considered. First, MC-PILCO is tested on a cart-pole system and compared to other policy gradient algorithms, namely PILCO and Black-DROPS. In the same environment, we tested the capability of MC-PILCO to handle multimodal probability distributions. Second, MC-PILCO learns a controller in joint space of a UR5 robot arm, considered as an example of a higher degrees of freedom (DoF) system.

2.6.1 Cart-pole: comparison with other algorithms

We tested PILCO [3], Black-DROPS [4], and MC-PILCO on the cart-pole system with the same setup described in Section 2.5. In MC-PILCO, we considered the cost function (2.16) with lengthscales ($l_\theta = 3, l_p = 1$), and adopted the SE kernel, as it is the one employed by the other algorithms. As done before, we used the common metric given by cost (2.17) to compare the obtained results. The observed cumulative costs are reported in Figure 2.8.

MC-PILCO achieved the best performance both in transitory and at convergence.

[3]PILCO code available at `http://mlg.eng.cam.ac.uk/pilco/`
[4]Black-DROPS code available at `https://github.com/resibots/blackdrops`

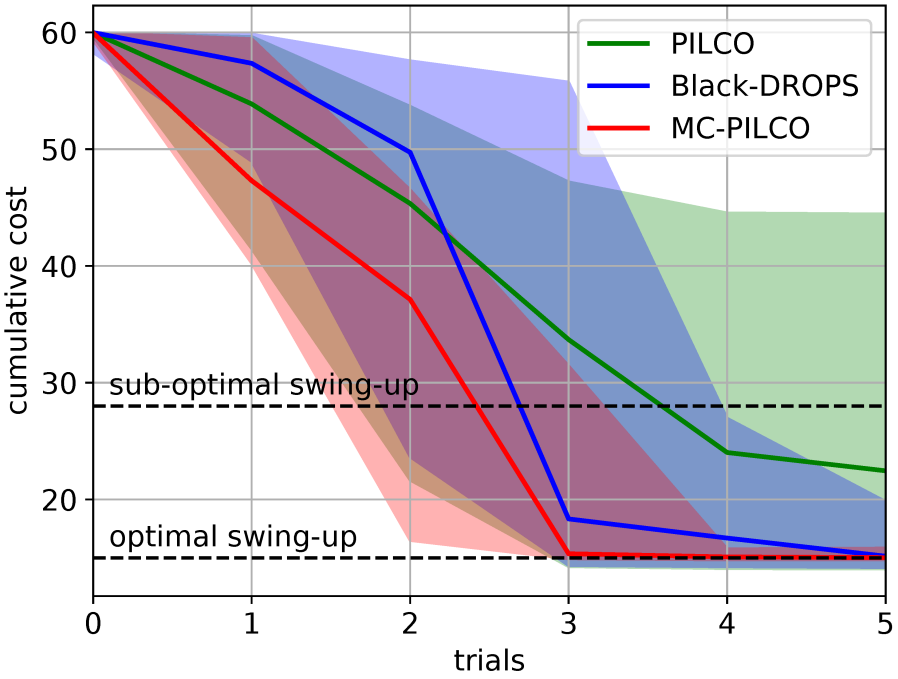


Figure 2.8: Median and confidence interval of the cumulative cost $c^{\text{pilco}}(\cdot)$ per trial obtained with PILCO, Black-DROPS and MC-PILCO (with *speed-integration* model, SE kernel, dropout activated, $l_\theta = 3$ and $l_p = 1$). Success rates are reported below.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
PILCO	2%	4%	20%	36%	42%
Black-DROPS	0%	4%	30%	68%	86%
MC-PILCO	0%	14%	78%	94%	100%

In fact, it learned how to swing up the cart-pole with a success rate of 100% by trial 5. In each and every trial, MC-PILCO obtained cumulative costs with lower median and less variability. On the other hand, PILCO showed poor convergence properties, with a final success rate of only 42%. Black-DROPS outperforms PILCO, but it obtained worse results than MC-PILCO, with a success rate of only 86% at trial 5. Recall that MC-PILCO obtains even better performance when considering SE+P$^{(2)}$ kernel. Results in Table 2.2, rows 1-2-3-6-7, also show that policies learned with MC-PILCO are more precise in reaching the target.

2.6.2 Cart-pole: handling multimodal distributions

One of the main advantages of particle-based policy optimization is the capability to handle multimodal state evolution. This is not possible when applying methods based on moment matching, such as PILCO. We verified this advantage by applying both

		e_p [m]	e_θ [rad]
1	S.I. SE+P$^{(2)}$ (3,1) drop. on	0.008 ± 0.003	0.011 ± 0.004
2	S.I. SP (3,1) drop. on	0.008 ± 0.003	0.011 ± 0.005
3	S.I. SE (3,1) drop. on	0.010 ± 0.005	0.011 ± 0.005
4	S.I. SE (0.75,0.25) drop. off	0.016 ± 0.009	0.012 ± 0.008
5	S.I. SE (3,1) drop. off	0.019 ± 0.014	0.015 ± 0.009
6	F.S. SE (3,1) drop. on	0.011 ± 0.005	0.011 ± 0.005
7	Black-DROPS	0.025 ± 0.011	0.033 ± 0.019
8	PILCO	0.027 ± 0.012	0.045 ± 0.019

Table 2.2: Average distances from the target states ($p_t = 0$ and $\theta_t = \pm\pi$) obtained during the last second of interaction with the cart-pole by the successful policies learned by PILCO, Black-DROPS and the various MC-PILCO configurations analyzed in Section 2.5. Different configurations are labeled reporting the adopted dynamical model structure (*speed-integration*, S.I., or *full-state*, F.S.), kernel function, cost length-scales, and if dropout was used or not. Values are reported as mean ± standard deviation, calculated over the total number of successful runs at trial 5.

PILCO and MC-PILCO to the simulated cart-pole system, when considering a very high variance on the initial cart position, $\sigma_p^2 = 0.5$, which corresponds to have unknown cart's initial position (but limited within a reasonable range). The aim is to be in a situation in which the policy has to solve the task regardless of the initial conditions and needs to have a bimodal behaviour in order to be optimal. Note that the situation described could be relevant in several real applications. We kept the same setup used in previous cart-pole experiments, changing the initial state distribution to a zero mean Gaussian with covariance matrix $\text{diag}([0.5, 10^{-4}, 10^{-4}, 10^{-4}])$). MC-PILCO optimizes the cost in (2.16) with length-scales ($l_\theta = 3, l_p = 1$). We tested the policies learned by the two algorithms starting from nine different cart initial positions (-2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2 [m]). In Section 2.6.1, we observed that PILCO struggles to consistently converge to a solution and the high variance in the initial conditions accentuates this issue. Nevertheless, in order to make the comparison possible, we cherry-picked a random seed for which PILCO converged to a solution in this particular scenario. In Figure 2.9, we show the results of the experiment. MC-PILCO is able to handle the initial high variance. It learned a bimodal policy that pushes the cart in two opposite directions, depending on the cart's initial position, and stabilizes the system in all the experiments. On the contrary, PILCO's policy is not able to control the cart-pole for all the tested starting conditions. Its strategy is always to push the cart in the same direction, and it cannot stabilize the system when the cart starts far away from the zero position. The state evolution under MC-PILCO's policy is bimodal, while PILCO cannot find this type of solutions because of the unimodal approximation enforced by moment matching.

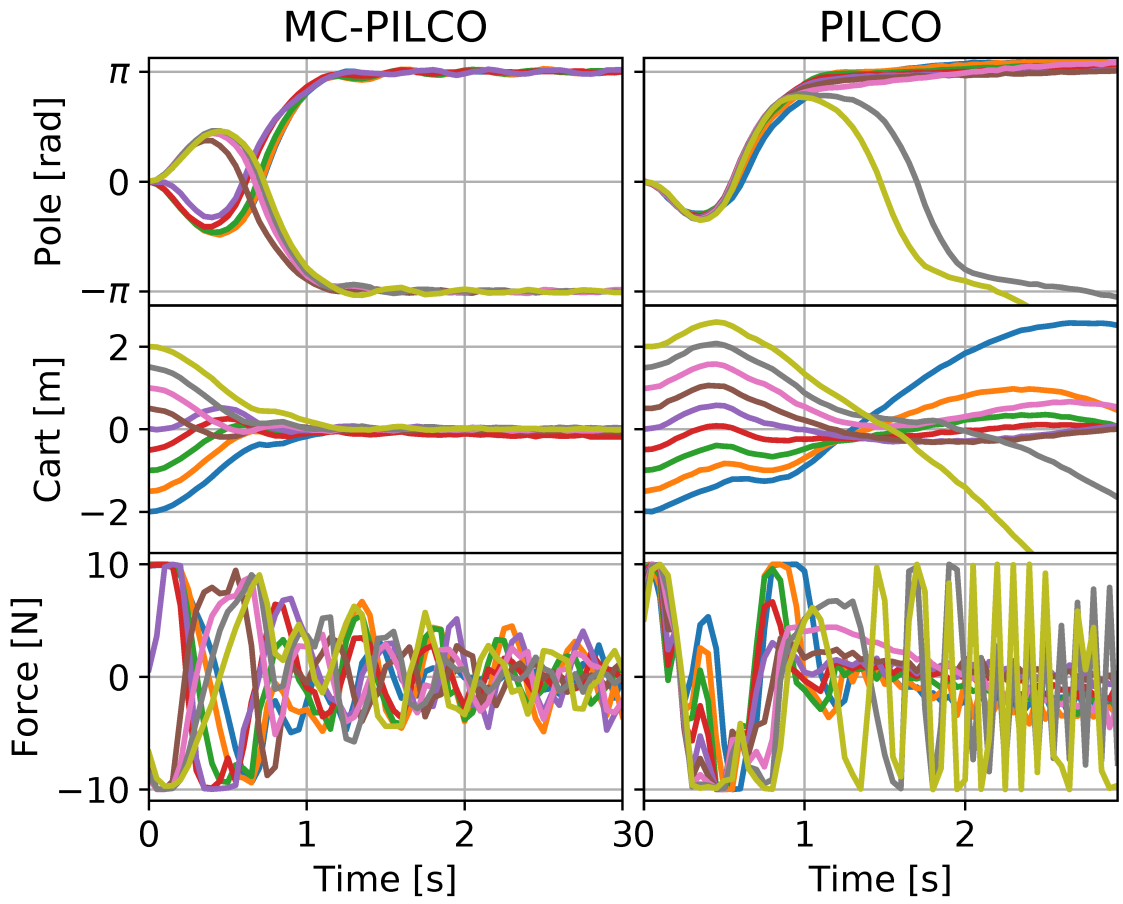


Figure 2.9: (Left) MC-PILCO policy applied to the cart-pole system starting from nine different sparse cart initial positions, namely: -2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2 [m]. The policy is able to complete the task in all cases, pushing the cart in different directions depending on its initial condition. (Right) PILCO policy applied starting from the same cart initial positions. This policy struggles to adapt to different starting conditions, and it cannot swing up the cart-pole when starting from the initial positions further away from zero.

In this example, we have seen that a multimodal state evolution could be the optimal solution, when starting from a unimodal state distribution with high variance, due to dependencies on initial conditions. In other cases, multimodality could be directly enforced by the presence of multiple possible initial conditions that would be badly modeled with a single unimodal distribution. MC-PILCO can handle all these situations thanks to its particle-based method for long-term predictions. Similar results were obtained when considering bimodal initial distributions.

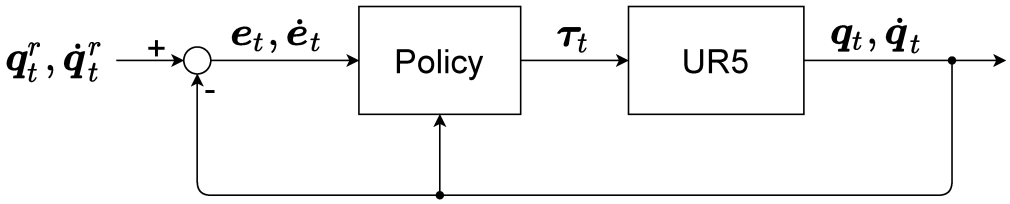


Figure 2.11: Joint-space control scheme for UR5 robotic arm.

2.6.3 UR5 joint-space controller: high DoF application

The objective of this experiment is to test MC-PILCO in a more complex system with higher DoF. We used MC-PILCO to learn a joint-space controller for a UR5, a robotic manipulator with 6 DoF (see Figure 2.10), simulated in MuJoCo [93]. Let the state at time t be $\boldsymbol{x}_t = [\boldsymbol{q}_t^T, \dot{\boldsymbol{q}}_t^T]^T$, where $\boldsymbol{q}_t$,$\dot{\boldsymbol{q}}_t \in \mathbb{R}^6$ are joint angles and velocities, respectively. The objective for the policy $\pi_{\boldsymbol{\theta}}$ is to control the torques $\boldsymbol{\tau}_t$ in order to follow a desired trajectory $(\boldsymbol{q}_t^r, \dot{\boldsymbol{q}}_t^r)$ for $t = 1, \dots, T$. Let $\boldsymbol{e}_t = \boldsymbol{q}_t^r - \boldsymbol{q}_t, \dot{\boldsymbol{e}}_t = \dot{\boldsymbol{q}}_t^r - \dot{\boldsymbol{q}}_t$ be position and velocity errors at time t, respectively. The policy is a multi-output *squashed-RBF-network* with $n_b = 400$ Gaussian basis functions and $u_{max} = 1$ [N·m] for all the joints, that maps states and errors into torques, $\pi_{\boldsymbol{\theta}} : \boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{e}_t, \dot{\boldsymbol{e}}_t \mapsto \boldsymbol{\tau}_t$. The control scheme is represented in Figure 2.11.

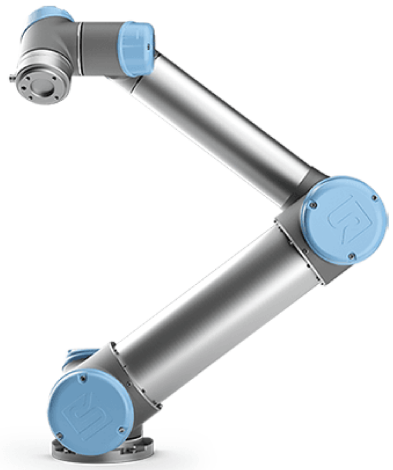


Figure 2.10: UR5 robot.

In this experiment, we considered a control horizon of 4 seconds with a sampling time of 0.02 seconds. The reference trajectory has been calculated to make the end-effector to draw a circle in the X-Y operational space. The initial exploration, used to initialize the *speed-integration* dynamical model, is provided by a poorly-tuned PD controller. We used SE+P$^{(1)}$ kernels in the GP dynamical model. The GP reduction thresholds were set to 10^{-3}. We considered an extended version of the state $\boldsymbol{x}_t^* = [\dot{\boldsymbol{q}}_t, sin(\boldsymbol{q}_t), cos(\boldsymbol{q}_t)]$ for building both GP and policy inputs. $M = 200$ is the number of particles used for gradient estimation. The cost function considered is defined as,

$$c(\boldsymbol{x}_t) = 1 - \exp\left(-\left(\frac{||\boldsymbol{q}_t^r - \boldsymbol{q}_t||}{0.5}\right)^2 - \left(\frac{||\dot{\boldsymbol{q}}_t^r - \dot{\boldsymbol{q}}_t||}{1}\right)^2\right).$$

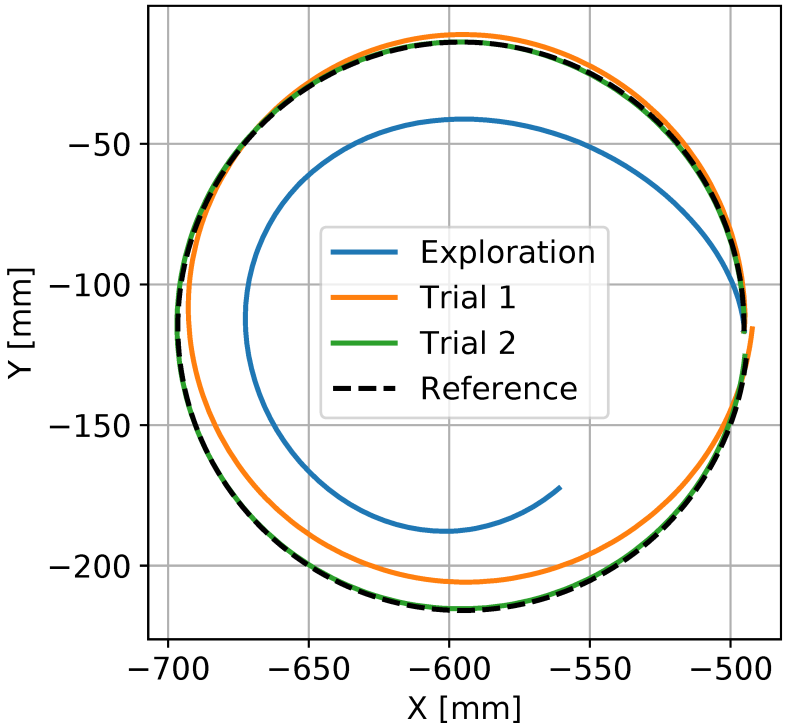


Figure 2.12: End-effector trajectories obtained in exploration and for each trial of policy learning together with the desired circle. Let $\boldsymbol{e}_{ee}$ be the error between the desired and the actual end-effector trajectories. In the Table below, we report, in millimeters, the maximum and mean errors (± standard deviation) at each trial.

	Exploration	Trial 1	Trial 2
mean($\boldsymbol{e}_{ee}$) [mm]	140.66±52.98	21.15±13.90	0.65±0.23
max($\boldsymbol{e}_{ee}$) [mm]	196.70	40.79	1.08

We assumed full state observability with measurements perturbed by a white noise with standard deviation of 10^{-3}. The initial state distribution is a Gaussian centered on $(\boldsymbol{q}_0^r, \dot{\boldsymbol{q}}_0^r)$ with standard deviation of 10^{-3}. Policy optimization parameters are the same reported in Table 2.1, with the exception of $n_s = 400$ and $\sigma_s = 0.05$, to enforce more restrictive exit conditions.

In Figure 2.12, we report the trajectory followed by the end-effector at each trial, together with the desired trajectory. MC-PILCO considerably improved the high tracking error obtained with the PD controller after only 2 trials (corresponding to 8 seconds of interaction with the system). The learned control policy followed the reference trajectory for the end-effector with a mean error of 0.65 ± 0.23 [mm] (confidence interval computed as standard deviation), and a maximum error of 1.08 [mm].

2.7 Conclusion

In this chapter, we have presented the MBRL algorithm MC-PILCO. The proposed framework uses GPs to derive a probabilistic model of the system dynamics, and updates the policy parameters through a gradient-based optimization; the optimization exploits the *reparameterization trick* and approximates the expected cumulative cost relying on a Monte Carlo approach. Compared to similar algorithms proposed in the past, the Monte Carlo approach worked by focusing on two aspects, that are (i) proper selection of the cost function, and (ii) introduction of exploration during the policy optimization through the use of dropout. We compared MC-PILCO with PILCO and Black-DROPS, that are two state-of-the-art GP-based MBRL algorithms. MC-PILCO outperforms both the algorithms, exhibiting better data-efficiency and asymptotic performance. The results obtained in simulation confirm the effectiveness of the proposed solution, and show the relevance of the two aforementioned aspects when optimizing the policy combining the *reparameterization trick* with particles-based approaches. Moreover, we explored two advantages due to the particles-based approximation w.r.t. the moment-matching adopted in PILCO, that are, the possibility of using structured kernels, such as polynomial and semi-parametrical kernels, and the ability of handling multimodal distributions. In particular, experimental results show that the use of structured kernels can increase data-efficiency, reducing the interaction-time required to learn the task.

Nevertheless MC-PILCO, as it is, can properly work only when full state observability is guaranteed. This is a common assumption when working in simulated environment, but in the real world, the majority of systems can only be observed through noisy partial measurements, from which the actual real state must be estimated. This issue has often been overlooked by previous MBRL methods, but it remains of critical importance for achieving a successful deployment of such techniques in the real world. In the next chapter, we will show how MC-PILCO strategy can be adapted to effectively work when only partial measurements of the state are available.

3

Policy Search for Partially Measurable Systems

One of the main obstacles that MBRL techniques must deal with in order to successfully work in real-world environments is the lack of full state measures characterizing the vast majority of real systems. For instance, in mechanical systems, the joint positions can be measured by means of proper sensors, e.g. encoders, whereas velocities often can only be estimated from the history of measured positions, unless more expensive velocity sensors (tachometers) are installed. We indicate such systems with the name of Partially Measurable Systems (PMS), characterized by a state that is observable, but with only some of its components that can be directly measured, while the rest must be estimated from measurements. Then, when dealing with PMS, it is necessary to design and implement state observers able to correctly estimate the full state from a series of partial measures. Additionally, in presence of a significant noise, it may be also necessary to filter the measurement signals.

In this regard it is worth mentioning the different characteristics of the estimation procedures that take place in the two different contexts of our interest: modeling and control. In the first case, it is possible to rely on offline techniques, intrinsically acausal, that exploits the complete history (past and future) of measurements to obtain an accurate estimation the true state of the system, i.e. the estimate of the state at time t depends both on past measurements, collected before t, and future measures, obtained after t. On the other hand, when trying to estimate the true state for control purposes,

we are restricted by a real-time constraint. The estimation procedure must be performed online. It is intrinsically casual and its computational time must be compatible with the control frequency, since the state estimator must be integrated inside the control loop. Typically, online estimates are affected by delays and they are generally less accurate than their offline counterparts, for causality reasons and real-time computation constraints.

Such concerns, in our opinion, have not been duly taken into account in most of previous works on MBRL, despite their critical importance in any possible real-world application. Usually algorithms are tested in simulated environments, where the state is assumed to be completely accessible. A correct understanding of the state estimation problem, for both modeling and control purposes, becomes of capital importance, in particular for MBRL approaches, which fully rely on the model predictions for updating the policy. For instance, the previous applications of MBRL algorithms on real systems, available in literature, did not carefully take into account the problems related to velocity estimation. In [35], PILCO was tested on a real cart-pole, but it was not mentioned specifically how velocities were derived. One could imagine that the same velocity estimates obtained inside the control loop were used for training the model. But, in this way, the policy would be optimized based on "wrong" velocity estimates (for reasons of causality, they would be inevitably affected by delays, if not other major discrepancies due to noise). Nevertheless, the task was completed successfully, but we can suppose that this result is mainly due to the fact that the optimized policy can rely on correct position predictions. On the other hand, in [36] and [19], PILCO and Black-DROPS, respectively, were used to learn from scratch how to control low-cost robot manipulators. In both cases, the state comprised only positions. Such choice can be effective only when the target behaviour does not involve fast dynamics, as in the case of the two aforementioned applications.

In this chapter, we present a modified version of the MC-PILCO algorithm of Chapter 2, specifically tailored to deal with the issues related with the application to real PMS. We label this different version MC-PILCO4PMS: *Monte Carlo Probabilistic Inference for Learning COntrol for Partially Measurable Systems*. In particular, MC-PILCO4PMS takes into account the differences between online and offline state estimators. The controller, i.e., the policy, works with the output of online state estimators which, due to noise and real-time computation constraints, might introduce significant delays and discrepancies w.r.t. to the offline estimates used for model learning. In this context, we verified that during policy optimization it is important to distinguish between the states generated by the models, which aim at describing the evolution of the real system state, and the states provided to the policy. Indeed, providing to the control policy the

model predictions corresponds to assuming to measure the system state directly, which, as mentioned before, is not possible in the real system. This incorrect assumption might compromise the effectiveness of the trained policy into the real system, due to the presence of distortions caused by the online state estimators. Hence, during policy optimization, from the evolution of the system state predicted by the GP models, we compute the estimates of the states observed by modeling both the measurement system and the online estimators used in the real system. Then we feed to the policy the estimates of the observed states. In this way, we aim at obtaining robustness w.r.t. the delays and distortions caused by online filtering. Thanks to the flexibility of our particle-based approach, it is possible to easily reproduce a wide variety of filters and state estimators, e.g., numerical differentiation, low-pass filters, Kalman filters, etc. The effectiveness of the proposed strategy has been tested in the case of mechanical systems both in simulation and also with two real systems, a Furuta pendulum and a ball-and-plate. The obtained performance confirms the importance of considering the presence of filters in the real system during policy optimization.

The rest of the chapter is structured as follows. In Section 3.1 we discuss more in details the problems related to state estimation for mechanical systems. In Section 3.2, we discuss the application of the proposed algorithm in such context. A proof of concept is proposed in Section 3.3. Experiments on a real Furuta pendulum and a ball-and-plate system are shown in Section 3.4.

3.1 State Estimation in Mechanical Systems

Consider a mechanical system with $d_{\boldsymbol{q}}$ DoF. Let $\boldsymbol{q}_t \in \mathbb{R}^{d_{\boldsymbol{q}}}$ indicate the vector of joint positions, while we denote the associated velocities with the symbol $\dot{\boldsymbol{q}}_t \in \mathbb{R}^{d_{\boldsymbol{q}}}$. We refer to the state of the mechanical system at time t with $\boldsymbol{x}_t \in \mathbb{R}^{d_{\boldsymbol{x}}}$, where $d_{\boldsymbol{x}} = 2d_{\boldsymbol{q}}$. Typically, the state is defined as $\boldsymbol{x}_t = [\boldsymbol{q}_t^T, \dot{\boldsymbol{q}}_t^T]^T$. We assume, as it is often true in reality, to have access only to noisy measurements of positions $\boldsymbol{q}_t$, while velocities $\dot{\boldsymbol{q}}_t$ must be estimated from position measures. Hence, mechanical systems perfectly fall into the PMS category, previously defined. Consequently, the $\dot{\boldsymbol{q}}_t$ elements are estimated starting from the history of $\boldsymbol{q}_t$ measurements through proper estimation procedures, possibly performing also denoising operations of $\boldsymbol{q}_t$ in case that the measurement noise is high. In particular, it is worth distinguishing between estimates computed online and estimates computed offline. The former are provided to the control policy to determine the system's control input, and they need to respect real-time constraints, namely, velocity estimates are causal and computations must be performed within a given interval. The latter, do not have to deal

with such constraints. As a consequence, offline estimates can be more accurate, taking into account acausal information and limiting delays and distortions.

As an example, here we analyze the velocity estimation problem in the simulated cart-pole environment considered in the previous chapter and described in details in Section 2.5. Remember that we indicate with p_t and θ_t, respectively, the cart position and pole angle at time step t. Differently from before, we assume to be able to directly measure only p_t and θ_t, and not also their respective velocities. We modeled a possible measurement system as an additive Gaussian i.i.d. noise with standard deviation $3 \cdot 10^{-3}$. In order to obtain reliable estimates of the velocities, samples were collected at 30 [Hz]. The online and offline state estimators implemented are the following:

- *Online Estimator*: Position measures are not filtered to avoid delays. Velocities are estimated by means of causal numerical differentiation, i.e. $\dot{\boldsymbol{q}}_t = (\boldsymbol{q}_t - \boldsymbol{q}_{t-1})/T_s$ (where T_s indicates the sampling time) followed by a first order low-pass filter with cutoff frequency 7.5 [Hz].

- *Offline Estimator*: Positions measures are filtered by applying, forward and backward, a second order low-pass filter with cutoff frequency 7.5 [Hz]. Velocities are then estimated by means of central difference, i.e. $\dot{\boldsymbol{q}}_t = (\boldsymbol{q}_{t+1} - \boldsymbol{q}_{t-1})/(2\ T_s)$.

Note that the combined filtering procedure performed offline has zero phase, resulting in no delays in the filtered signal. Obviously, this is an acausal data processing that can be performed only offline. It is worth mentioning that the estimators proposed are only one possible choice and several different filters could be adopted. Anyway the considerations that follow are still valid also for other filters and estimators.

In Figure 3.1, we compare the different velocities obtained online and offline with their true values, that are available from the simulation. It is clear how the online estimates are affected by a delay caused by the causal filtering process. This effect is inevitable and, in case the delay is significantly high, it must be carefully taken into account to avoid problems in the control loop. On the other hand, the offline estimate are far more accurate and present no delays with respect to the true velocities. Hence, these signals can be considered valid estimates and used for modeling purposes. Clearly, the predictive model cannot be build based upon the online estimates, being the latter afflicted by delays and distortions.

In the next section, we will carefully take into account these considerations in order to modify the MC-PILCO algorithm of Chapter 2 so that it may be applied to real PMS, correctly facing all the issues related to the state estimation.

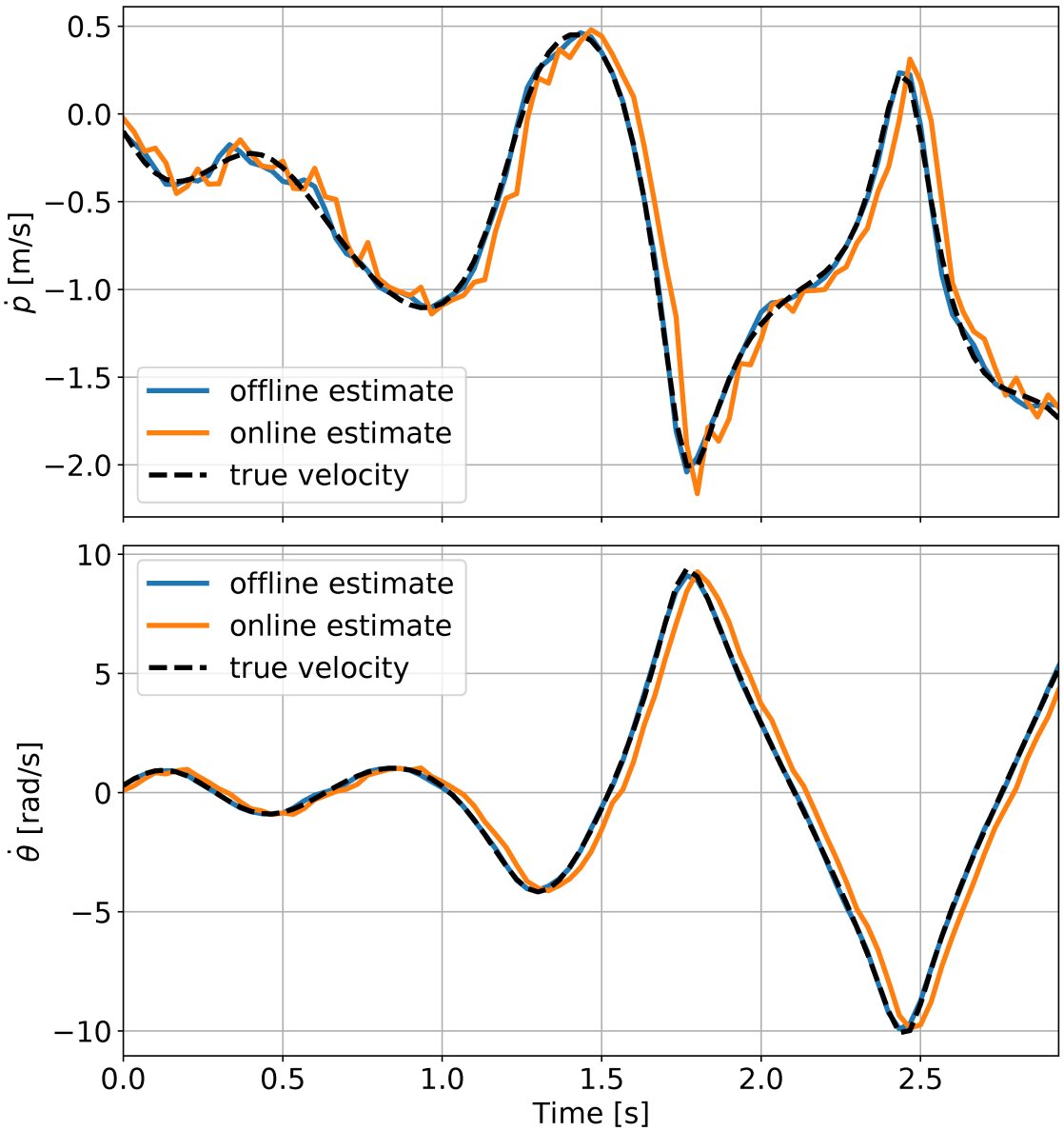


Figure 3.1: Comparison between offline and online velocity estimation in the cart-pole scenario. Also the true velocity available from the simulation is reported.

3.2 MC-PILCO for Partially Measurable Systems

In this section, we discuss the application of MC-PILCO to PMS, that, we remember, are systems whose state is observable, but only some of its components can be directly measured, while the rest must be estimated from measurements. For simplicity, we introduce the problem discussing the case of mechanical systems, previously presented in Section 3.1, but similar considerations can be done for any PMS. So, we present MC-PILCO for Partially Measurable Systems (MC-PILCO4PMS), a modified version of MC-PILCO, proposed to deal with such setups.

In this context, we verified that, during policy optimization, it is relevant to distinguish between the particle state predictions computed by the models and the data provided to the policy. Indeed, GPs should simulate the real system dynamics, independently of additional noise given by the sensing instrumentation, so they need to work with the most accurate estimates available; delays and distortions might compromise the accuracy of long-term predictions. On the other hand, providing to the policy directly the particles' states computed with the GPs during policy optimization, correspond to train the policy assuming available access to the system state, which, as mentioned before, is not possible for PMS. Indeed, considerable discrepancies between the particles' states and the state estimates computed online during the policy application to the real system might compromise the effectiveness of the policy. Thus, the policy optimization process must be "aware" of the online estimator implemented in the control loop, in order to train the policy to cope with the presence of delays and differences w.r.t. the true state of the system. This approach differs from standard MBRL approaches where, typically, the effects of the online state estimators are not considered during training and full-state measurability is assumed.

To deal with the above issues, we introduce MC-PILCO4PMS a modified version of MC-PILCO. With respect to the algorithm described in Chapter 2, we propose the two following additions:

- *Offline estimation of GP training data*: the GPs of the predictive model (be it the *full-state* or the *speed-integration* model described in Sections 2.1.2 and 2.3.1, respectively) are now trained based on state estimates computed offline from the series of noisy measurements collected during previous interactions with the system.
- *Simulation of the online estimator during policy optimization*: the policy parameters are updated based on particles simulations that replicates the presence of noisy sensors and online state estimator inside the control loop. In this way, the policy is trained to work properly in the real scenario.

In the remainder of this section, we present the offline estimation techniques adopted in the final experiments and describe in details the new simulation of the online estimator during the particle-based policy optimization phase.

3.2.1 Offline estimation of the GP training data

The collected measurements are processed by offline estimation techniques to obtain the training data for the GP models. In particular, in our experiments with real systems, we considered two options:

- Computation of the velocities by central difference, i.e., $\dot{\boldsymbol{q}}_t = (\boldsymbol{q}_{t+1} - \boldsymbol{q}_{t-1})/(2T_s)$, where T_s is the sampling time. This technique can be used only when the measurement noise is limited, otherwise the $\dot{\boldsymbol{q}}$ estimates might be too noisy. In such cases it is appropriate to pre-filter the measures as previously seen in Section 3.1.

- Estimation of the state by means of Kalman smoother [38], with state-space model given by the general equations relating positions, velocities, and accelerations. The advantage of this technique is that it exploits the correlation between positions and velocities, increasing regularization.

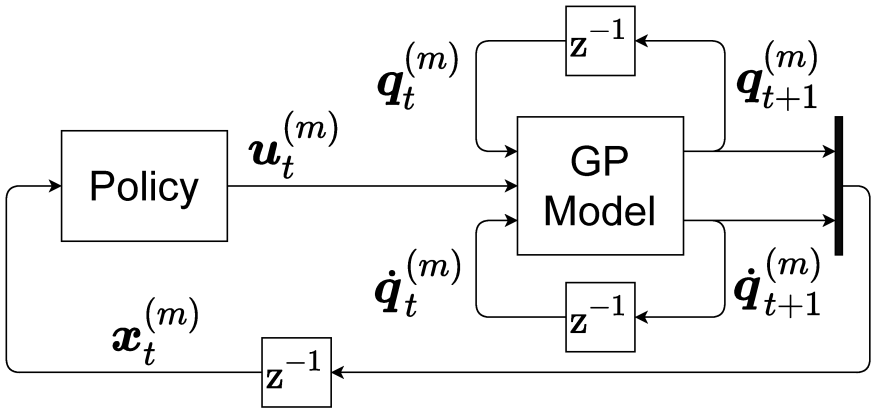
3.2.2 Simulation of the online estimator during policy optimization

Standard MC-PILCO algorithm updates the policy parameters following a gradient estimated from some particle trajectories, that are simulated by the trained predictive model. The simulated particles aim to approximate the true state evolution. In the context of PMS, it would not be correct to use the state particles as input of the control policy, since we already specified that the policy works on state estimates computed online. Hence, in MC-PILCO4PMS, during the policy optimization phase, we simulate also the presence of the measurement system and the online estimator. The state fed to the policy, denoted $\bar{\boldsymbol{x}}_t$, is computed to resemble the state that will be estimated online.

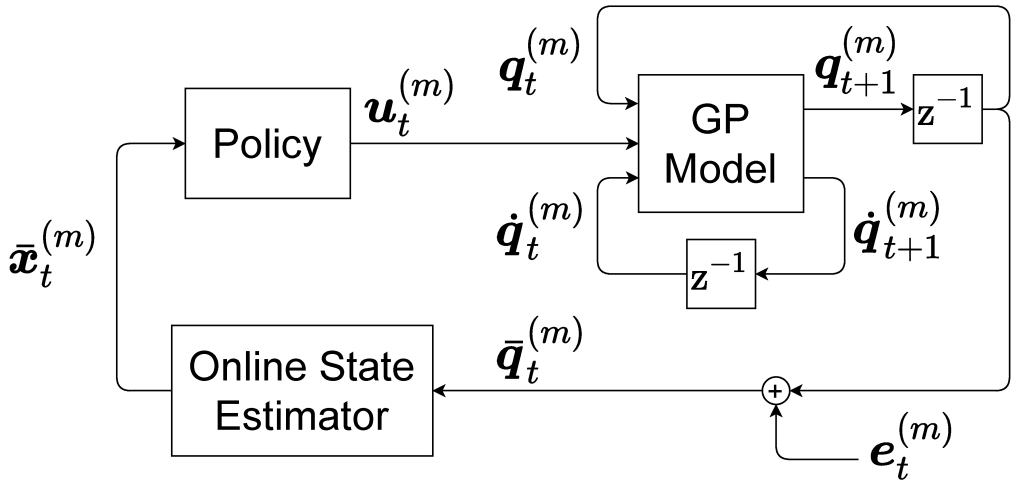
Recalling the notation of Section 2.4.2, let us denote with $\boldsymbol{x}_t^{(m)} = [\boldsymbol{q}_t^{(m)\,T}, \dot{\boldsymbol{q}}_t^{(m)\,T}]^T$ the m-th particle at time step t, where we explicitly indicated the position and velocity components. Thus, we simulate the presence of the measurement system by corrupting only the particle's position with a fictitious Gaussian i.i.d. noise $\boldsymbol{e}_t^{(m)} \sim \mathcal{N}(0, \mathrm{diag}([\sigma_z^{(1)} \dots \sigma_z^{(d_{\boldsymbol{q}})}]))$, obtaining a simulated measure of the m-th particle $\bar{\boldsymbol{q}}_t^{(m)}$ as

$$\bar{\boldsymbol{q}}_t^{(m)} = \boldsymbol{q}_t^{(m)} + \boldsymbol{e}_t^{(m)}, \tag{3.1}$$

The $\sigma_z^{(i)}$s values must be tuned in accordance with the properties of the measurement



(a) MC-PILCO



(b) MC-PILCO4PMS

Figure 3.2: Block schemes illustrating particles generation in MC-PILCO (left) and MC-PILCO4PMS (right).

system, e.g., the accuracy of the encoders.

The simulated online state estimate $\bar{\boldsymbol{x}}_t^{(m)}$ is then computed as

$$\bar{\boldsymbol{x}}_t^{(m)} = \varphi\left(\bar{\boldsymbol{q}}_t^{(m)} \dots \bar{\boldsymbol{q}}_{t-m_q}^{(m)}, \bar{\boldsymbol{x}}_{t-1}^{(m)} \dots \bar{\boldsymbol{x}}_{t-1-m_z}^{(m)}\right),$$

where φ denotes the online state estimator function, with memory m_q and m_z, implemented in the system. Finally, the control input associated to the m-th particle is computed as $\pi_{\boldsymbol{\theta}}(\bar{\boldsymbol{x}}_t^{(m)})$, instead than $\pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t^{(m)})$, as it would be in standard MC-PILCO. Differences in particles generation between MC-PILCO and MC-PILCO4PMS are summed up in the block scheme reported in Figure 3.2. Finally, MC-PILCO4PMS is summarized in pseudo-code in Algorithm 2.

Algorithm 2: MC-PILCO4PMS

init policy $\pi_{\boldsymbol{\theta}}(\cdot)$, cost $c(\cdot)$, kernel $k(\cdot,\cdot)$, maximum optimization steps N_{opt}, number of particles M, learning rate α_{lr}, min. learning rate $\alpha_{lr_{min}}$, dropout probability p_d, dropout probability reduction Δ_{p_d} and other monitoring signal parameters: σ_s, λ_s, n_s, offline state estimator, online state estimator.
Apply exploratory control actions to system and collect data;
Compute offline state estimates and initialize GP training data-set;
while *task not learned* **do**
  1) Model Learning:
  Train GP models based on the offline estimated data-set;
  2) Policy Update:
  Initialize monitoring signal $s_0 = 0$;
  for $j = 1...N_{opt}$ **do**
    Simulate M particles rollouts with GP models and current policy $\pi_{\boldsymbol{\theta}_j}(\cdot)$ considering the presence of the online state estimator as in Section 3.2.2;
    Compute $\hat{J}(\boldsymbol{\theta}_j)$ from particles (2.11);
    Compute $\nabla_{\boldsymbol{\theta}}\hat{J}(\boldsymbol{\theta}_j)$ through *backpropagation*;
    $\pi_{\boldsymbol{\theta}_{j+1}}(\cdot) \leftarrow$ Gradient-based policy update (e.g., Adam step);
    Update monitoring signal s_j with (2.12);
    if (2.13) *is True* **then**
      Update p_d, α_{lr} and σ_s with (2.14);
    end
    if (2.15) *is False* **then**
      break;
    end
  end
  3) Policy Execution:
  Apply updated policy to system and collect data;
  Compute offline state estimates and update GP training data-set;
end
return trained policy, learned GP model;

3.3 Proof of Concept

Here, we test the relevance of modeling the presence of online estimators using the PMS setup described in Section 3.1 for the simulated cart-pole system.

To verify the effectiveness of the MC-PILCO4PMS strategy, two policy functions were trained. The first policy is obtained with MC-PILCO by neglecting the presence of online filtering during policy optimization and assuming direct access to the state predicted by the model. On the contrary, the second policy is trained with MC-PILCO4PMS, which models the presence of the online estimators as previously described in Section 3.2.2. In both cases we equipped the GPs with SE kernel functions. Exploration data were collected applying a sequence of control inputs derived from a sum of ten sine functions with random amplitudes and frequencies. To avoid dependencies on initial conditions, such as policy initialization and exploration data, we fixed the same random seed in both experiments. In Figure 3.3, we report the results obtained in 400 runs by the policies learned after 5 trials, on both the trained model (MODEL plot column) and the actual cart-pole (SYSTEM plot column). Even though the two policies perform similarly when applied to the learned models, which is all that can be tested offline, the results obtained by testing the policies in the cart-pole system are significantly different. The MC-PILCO4PMS policy optimized with modeling the presence of online filtering solves the task in all 400 attempts. In contrast, the MC-PILCO policy is not able to solve the task in the system, due to delays and discrepancies introduced by the online filter and not considered during policy optimization.

We believe that these considerations on how to manipulate the data during model learning and policy optimization might be beneficial for other MBRL algorithms different from MC-PILCO.

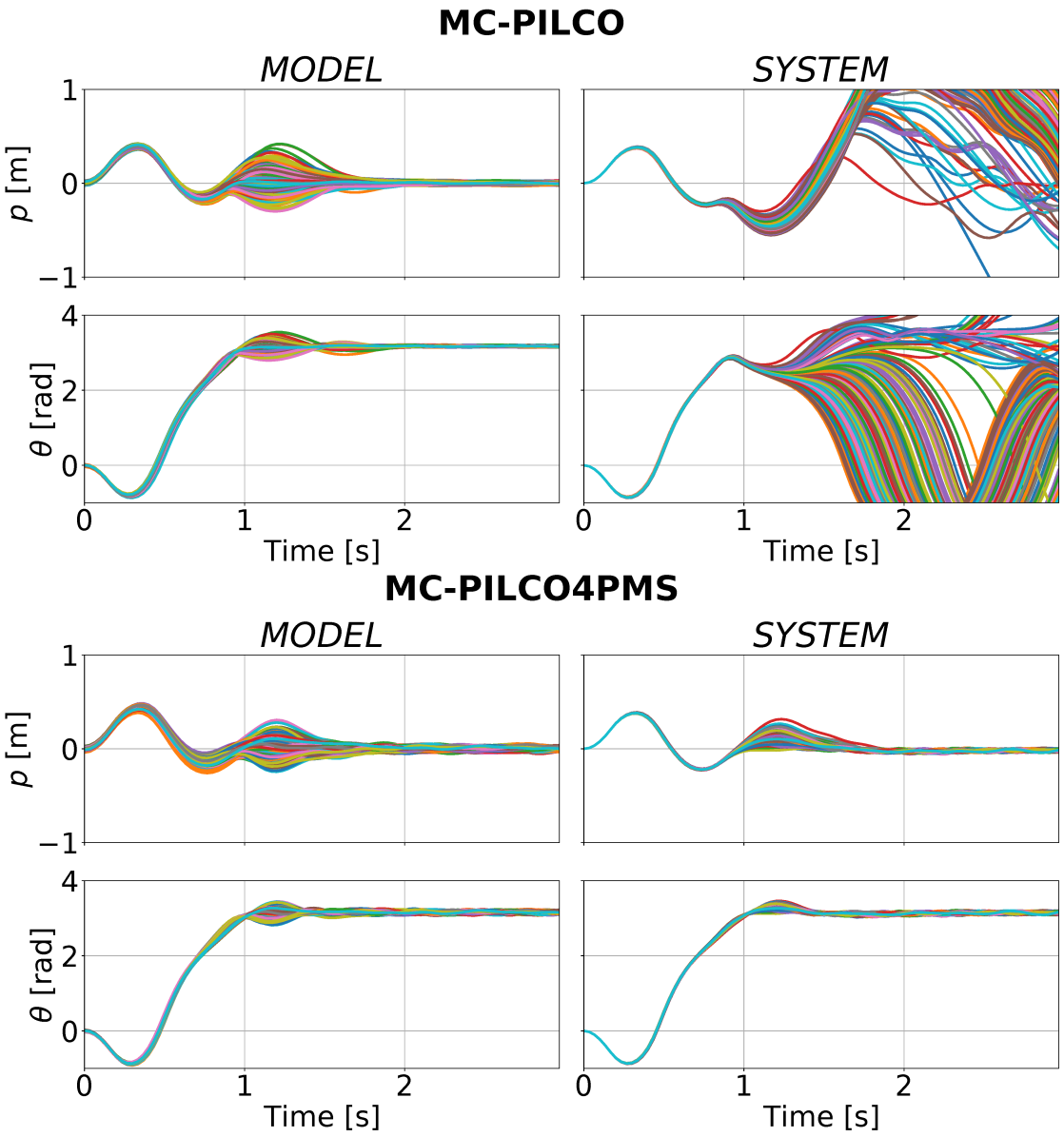


Figure 3.3: Comparison of 400 simulated particle rollouts (MODEL column) and the trajectories performed applying repetitively the policy 400 times in the system (SYSTEM column) on the simulated cart-pole. Results obtained without simulating online filtering are on the top plots (MC-PILCO), while the ones obtained considering the low-pass filters are on the bottom (MC-PILCO4PMS). The plots refer to the policy learned after 5 trials. It is clear how it is fundamental to model the presence of the online state estimator during policy optimization to obtain robust results on the actual system.

3.4 Experiments with Real Systems

In this section, we test MC-PILCO4PMS when applied to real systems, considering the concepts introduced in Section 3.2. In particular, we experimented on two benchmark systems: a Furuta pendulum, and a ball-and-plate (both represented in Figure 3.4)[1].

Figure 3.4: (Left) Furuta pendulum used in the experiment while being controlled in the upward equilibrium point by the learned policy. (Right) Ball-and-plate system.

3.4.1 Furuta pendulum

The Furuta pendulum [18] is a popular benchmark used in nonlinear control and reinforcement learning. The system is composed of two revolute joints and three links. The first link, called base, is fixed and perpendicular to the ground. The second link, called arm, rotates parallel to the ground, while the rotation axis of the last link, the pendulum, is parallel to the principal axis of the second link, see Figure 3.4 (left). The Furuta pendulum is an under-actuated system as only the first joint is actuated. In particular, in our system, the horizontal joint is actuated by a DC servomotor, and the two angles are measured by optical encoders with 4096 PPR. The control variable is the motor voltage.

Let the state at time step t be $\boldsymbol{x}_t = [\theta_t^h, \dot{\theta}_t^h, \theta_t^v, \dot{\theta}_t^v]^T$, where θ_t^h is the angle of the horizontal joint and θ_t^v the angle of the vertical joint attached to the pendulum. The

[1] A video of both experiments is available at *https://youtu.be/-73hmZYaHA*.

objective is to learn a controller able to swing-up the pendulum and stabilize it in the upwards equilibrium ($\theta_t^v = \pm\pi$ [rad]) with $\theta_t^h = 0$ [rad]. The trial length is 3 seconds with a sampling frequency of 30 [Hz]. The cost function is defined as

$$c(\boldsymbol{x}_t) = 1 - \exp\left(-\left(\frac{\theta_t^h}{2}\right)^2 - \left(\frac{|\theta_t^v| - \pi}{2}\right)^2\right) + c_b(\boldsymbol{x}_t), \tag{3.2}$$

with

$$c_b(\boldsymbol{x}_t) = \frac{1}{1 + \exp\left(-10\left(-\frac{3}{4}\pi - \theta_t^h\right)\right)} + \frac{1}{1 + \exp\left(-10\left(\theta_t^h - \frac{3}{4}\pi\right)\right)}.$$

The first part of the function in (3.2) aims at driving the two angles towards $\theta_t^h = 0$ and $\theta_t^v = \pm\pi$, while $c_b(\boldsymbol{x}_t)$ penalizes solutions where $\theta_t^h \leq -\frac{3}{4}\pi$ or $\theta_t^h \geq \frac{3}{4}\pi$. We set those boundaries to avoid the risk of damaging the system if the horizontal joint rotates too much. Offline estimates of velocities for the GP model have been computed by means of central difference. With regards to the online estimation, we used causal numerical differentiation, without filtering the position measurements, being that the noise affecting the system was not too much relevant. Instead of $\boldsymbol{x}_t$, we considered the extended state $\boldsymbol{x}_t^* = [\dot{\theta}_t^h, \dot{\theta}_t^v, sin(\theta_t^h), cos(\theta_t^h), sin(\theta_t^v), cos(\theta_t^v)]^T$ inside GP inputs. The policy is a *squashed-RBF-network* with $n_b = 200$ basis functions that receives as input $[(\theta_t^h - \theta_{t-1}^h)/T_s, (\theta_t^v - \theta_{t-1}^v)/T_s, sin(\theta_t^h), cos(\theta_t^h), sin(\theta_t^v), cos(\theta_t^v)]^T$, where T_s is the sampling time.

We used $M = 400$ particles to estimate the policy gradient from model predictions. The exploration trajectory has been obtained using as input a sum of ten cosine waves of random frequencies and same amplitudes. The initial state distribution is assumed to be $\mathcal{N}([0,0,0,0]^T, \text{diag}([5\cdot10^{-3}, 5\cdot10^{-3}, 5\cdot10^{-3}, 5\cdot10^{-3}])$. The policy optimization parameters used were the same described in Table 2.1 and GP reduction technique presented in Section 2.3.3 was employed, with thresholds set to 10^{-3}.

We solved the task using the three different kernel functions described in Section 2.3.2: *squared exponential* (SE), *squared exponential + polynomial of degree 2* (SE+P$^{(2)}$) and *semi-parametrical* (SP)[2]. In Figure 3.5, we show the resulting trajectories for each trial. MC-PILCO4PMS managed to learn how to swing up the Furuta pendulum in all cases. It succeeded at trial 6 with kernel SE, at trial 4 with kernel SE+P$^{(2)}$, and at trial 3 with SP kernel. These experimental results confirm the higher data efficiency of more structured kernels and the advantage that MC-PILCO4PMS offers by allowing any kind of kernel

[2]SP basis functions can be obtained by isolating, in each ODE defining Furuta pendulum laws of motion, all the linearly related state-dependent components. In particular, we have $\phi_{\dot{\theta}^h}(\boldsymbol{x}, u) = [(\dot{\theta}^v)^2 sin(\theta^v), \dot{\theta}^h\dot{\theta}^v sin(2\theta^v), \dot{\theta}^h, u]$ for the arm velocity GP, and $\phi_{\dot{\theta}^v}(\boldsymbol{x}, u) = [(\dot{\theta}^h)^2 sin(2\theta^v), \dot{\theta}^v, sin(\theta^v), u\, cos(\theta^v)]$ for the pendulum velocity GP.

function. Moreover, we can observe the effectiveness of the cost function (3.2) in keeping θ_t^h inside the desired boundaries most of the times, only during the first trial, the policy obtained with SE+P$^{(2)}$ slightly exceeded the lower limit. Hence, considering penalties similar to $c_b(\boldsymbol{x}_t)$ could be enough to handle soft constraints also in other scenarios.

3.4.2 Ball-and-plate

The ball-and-plate system is composed of a square plate that can be tilted in two orthogonal directions by means of two motors. On top of it, there is a camera to track the ball and measure its position on the plate. Let (b_t^x, b_t^y) be the position of the center of the ball along X-axis and Y-axis, while $\theta_t^{(1)}$ and $\theta_t^{(2)}$ are the angles of the two motors tilting the plate, at time t. So, the state of the system is defined as $\boldsymbol{x}_t = [b_t^x, b_t^y, \dot{b}_t^x, \dot{b}_t^y, \theta_t^{(1)}, \theta_t^{(2)}, \dot{\theta}_t^{(1)}, \dot{\theta}_t^{(2)}]^T$. The drivers of the motors allow only position control, and do not provide feedback about the motors angles. To keep track of the motor angles, we defined the control actions as the difference between two consecutive reference values sent to the motor controllers, and we limited the maximum input to a sufficiently small value, such that the motor controllers are able to reach the target angle within the sampling time. Then, in first approximation, the reference angles and the motor angles coincide, and we have $u_t^{(1)} = \theta_{t+1}^{(1)} - \theta_t^{(1)}$ and $u_t^{(2)} = \theta_{t+1}^{(2)} - \theta_t^{(2)}$. The objective of the experiment is to learn how to control the motor angles in order to stabilize the ball around the center of the plate. Notice that the control task, with the given definition of inputs, is particularly difficult because the policy must learn to act in advance, and not only react to changes in the ball position. The cost function is defined as

$$c(\boldsymbol{x}_t) = 1 - \exp\left(-\left(\frac{b_t^x}{0.15}\right)^2 - \left(\frac{b_t^y}{0.15}\right)^2 - \left(\theta_t^{(1)}\right)^2 - \left(\theta_t^{(2)}\right)^2\right). \tag{3.3}$$

The trial length is 3 seconds, with a sampling frequency of 30 [Hz]. Measurements provided by the camera are very noisy, and cannot be used directly to estimate velocities from positions. We used a Kalman smoother for the offline filtering of ball positions b_t^x, b_t^y and associated velocities $\dot{b}_t^x, \dot{b}_t^y$. In the control loop, instead, we used a Kalman filter [45] to estimate online the ball state from noisy measures of positions. When simulating the online estimator during policy optimization, we tried both to perturb and to not perturb the positions of the predicted particles with some additive noise. We obtained similar performance in the two cases, this result may be due to the fact that the Kalman filter is able to effectively filter out the white noise added to particles. Concerning the model, we need to learn only two GPs predicting the evolution of the ball velocity because we directly control motor angles, hence, their evolution is assumed deterministic. GP inputs,

$\tilde{\boldsymbol{x}}_t$, include an extended version of the state, where angles have been replaced by their sines and cosines, and motor angular velocities have been estimated with causal numerical differentiation, hence $\tilde{\boldsymbol{x}}_t = [b_t^x, b_t^y, \dot{b}_t^x, \dot{b}_t^y, sin(\theta_t^{(1)}), cos(\theta_t^{(1)}), sin(\theta_t^{(2)}), cos(\theta_t^{(2)}), (\theta_t^{(1)} - \theta_{t-1}^{(1)})/T_s, (\theta_t^{(2)} - \theta_{t-1}^{(2)})/T_s, u_t^{(1)}, u_t^{(2)}]^T$ (T_s is the sampling time). We used the SE+P$^{(1)}$ kernel (2.9), where the linear part is acting only on a subset of the inputs, $\tilde{\boldsymbol{x}}_t^{lin} = [sin(\theta_t^{(1)}), sin(\theta_t^{(2)}), cos(\theta_t^{(1)}), cos(\theta_t^{(2)}), u_t^{(1)}, u_t^{(2)}]$. The GP reduction threshold was decreased to 10^{-4} because of the small distances that the ball can cover in a time step.

We considered $M = 400$ particles for policy gradient estimation. The policy is a multi-output *squashed-RBF-network*, with $n_b = 400$ basis functions, that receives in input $(b_t^x, b_t^y, \dot{b}_t^x, \dot{b}_t^y, \theta_t^{(1)}, \theta_{t-1}^{(1)}, \theta_t^{(2)}, \theta_{t-1}^{(2)})$, where the state of the ball is obtained through the Kalman filter; maximum angle displacement is $u_{max} = 4$ [deg] for both motors. The policy optimization parameters used were the same described in Table 2.1, with the difference that we used $\alpha_{lr} = 0.006$ as initial learning rate. The reduction of the learning rate is related to the use of small length-scales in the cost function, that are necessary to cope with the small range of movement of the ball. For the same reason, we set also $\alpha_{lr_{min}} = 0.0015$ and $\sigma_s = 0.05$. Initial exploration is given by two different trials, in which the control signals are two triangular waves perturbed by white noise. Mostly during exploration and initial trials, the ball might touch the borders of the plate. In those cases, we kept data up to the collision instant. A peculiarity of this experiment in comparison to the others seen before is a wide range of initial conditions. In fact, the ball could be positioned anywhere on the plate's surface, and the policy must control it to the center. The initial distribution of b_0^x and b_0^y is a uniform $\mathcal{U}(-0.15, 0.15)$, which covers almost the entire surface (the plate is a square with sides of about 0.20 [m]). For the other state components, $\theta_t^{(1)}$ and $\theta_t^{(2)}$, we assumed tighter initial distributions $\mathcal{U}(-10^{-6}, 10^{-6})$.

MC-PILCO4PMS managed to learn a policy able to control the ball around the center starting from any initial position after the third trial, 11.33 seconds of interaction with the system. Figure 3.6 shows the ball trajectories observed during the initial explorations and the attempted policy trials. Note how the initial position of the ball varies in each trial, in this way we were able to provide MC-PILCO4PMS with data from different areas of the plate. We tested the final policy starting from ten different points in order to verify the ability of the learned controller to stabilize the ball in the center of the plate from different positions, the resulting trajectories are reported in Figure 3.7. The mean steady-state error, i.e. the average distance of the final ball position from the center observed in the ten trials, was 0.0099 [m], while the maximum measured error was 0.0149 [m], which is lower than the radius of the ball, that is 0.016 [m].

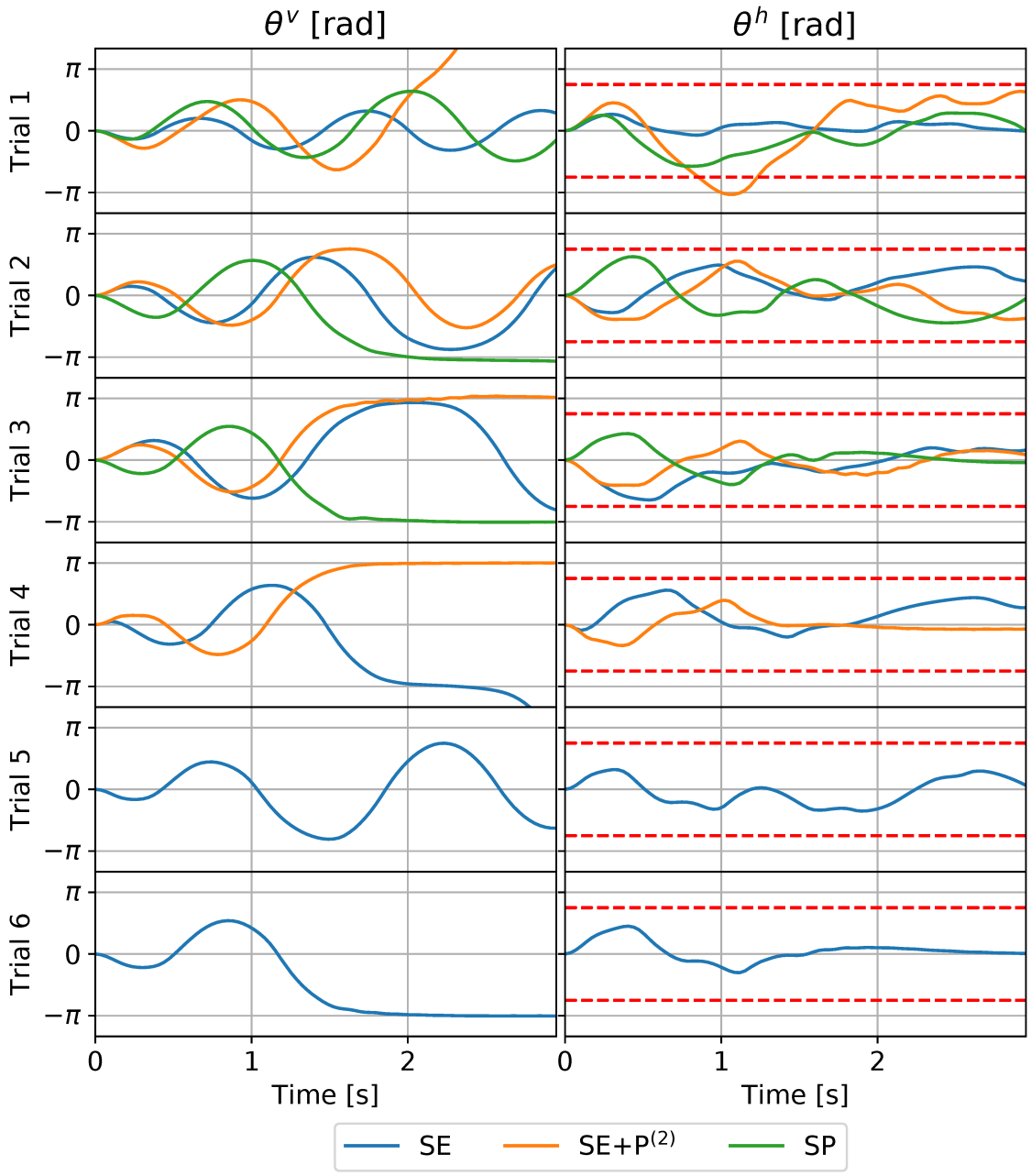


Figure 3.5: (Left) Pendulum angle's trajectories for each trial. (Right) Horizontal joint angle's trajectories for each trial. For all the kernels, the angles are plotted up to the trial that solved the task. In the θ^h plots we reported also the boundaries set in the cost function.

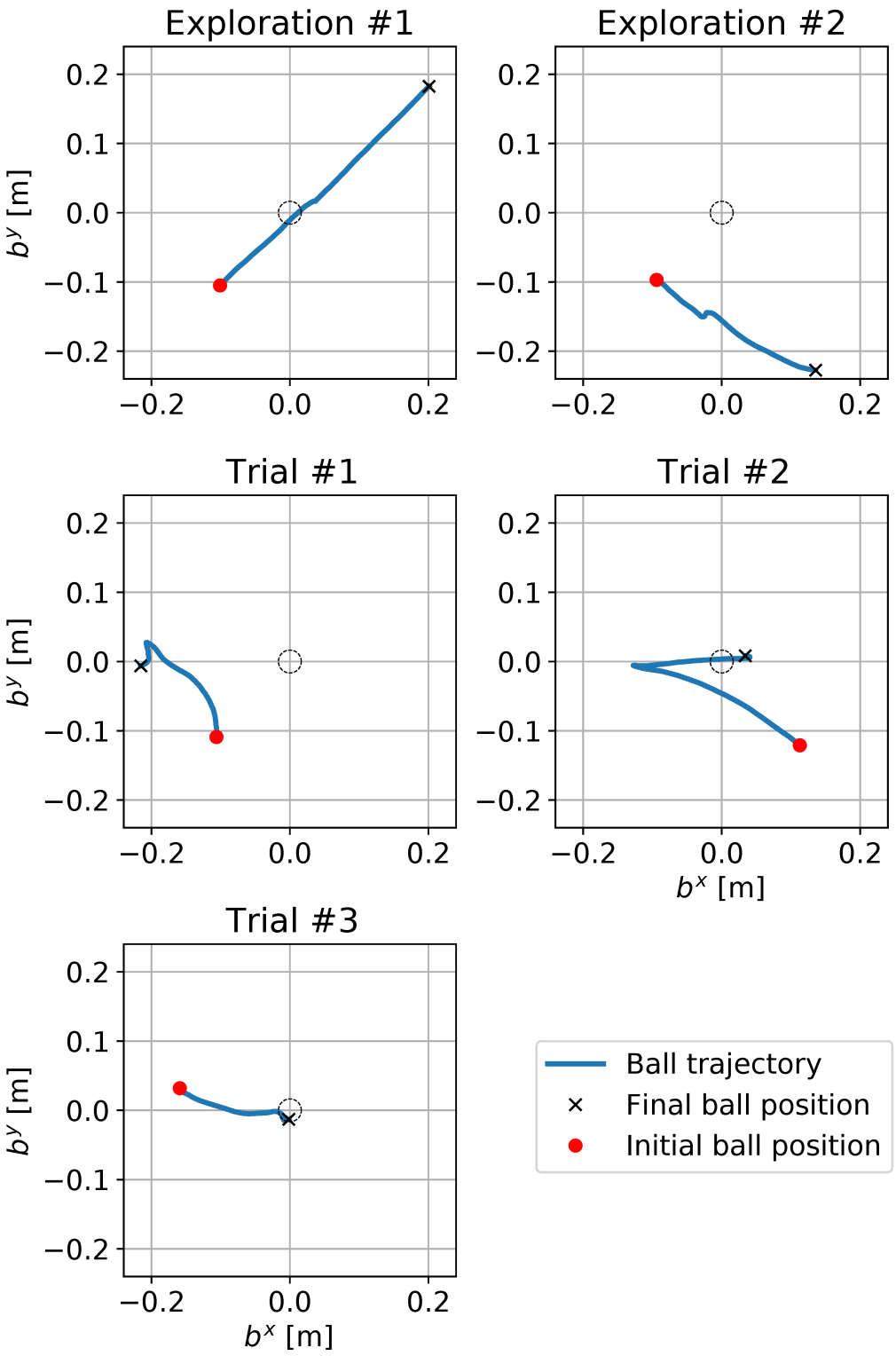


Figure 3.6: Ball trajectories observed during initial explorations and the trials attempted by MC-PILCO4PMS. Initial and final ball positions are indicated as a red circle and a black cross, respectively. The dashed circles are centered in the target positions and have the same diameter of the used ball.

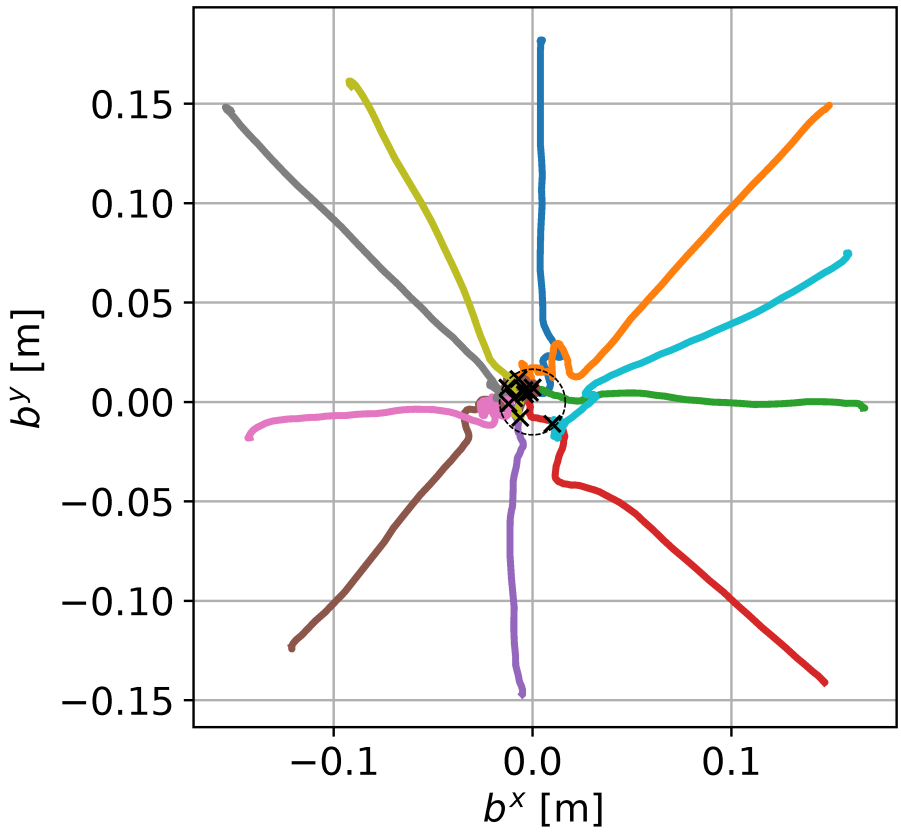


Figure 3.7: Ten different ball trajectories obtained under the final policy learned by MC-PILCO4PMS. Steady-state positions are marked with black crosses. The dashed circle is centered in the target position and has the same diameter of the used ball.

3.5 Conclusion

In this chapter, we analyzed common problems arising when trying to apply MBRL to real systems. In particular, we focused on systems with partially measurable states, which are particularly relevant in real applications. In this context, we proposed a modified version of the algorithm of Chapter 2 called MC-PILCO4PMS, where we verified the importance of taking into account the presence of the state estimators during policy optimization in the case of mechanical systems. Results have been validated in a simulated cart-pole swing-up scenario, and also on real systems, such as a Furuta pendulum and a ball-and-plate setup. Such considerations could be extended also to other partially measurable systems, and are not limited only to the case of velocity estimation.

The issues regarding the impossibility of measuring directly the velocity states tackled in MC-PILCO4PMS could be further analyzed by considering a Derivative-Free framework [27], that entails the presence of a predictive model based only on the history of positions. In the next chapter we will modify the MC-PILCO setting to work in such framework, circumventing totally, in this way, the necessity of performing velocity estimation.

4

Derivative-Free Model-Based Policy Search

In the previous chapter, we have seen that the application of MBRL methods to real-world environments is affected by a major problem: the full state of a real system is often only partially measurable. In particular, in the case of mechanical systems, the joint positions can be measured by means of proper sensors, e.g. encoders, whereas velocities often can only be estimated from the history of measured positions. The MC-PILCO4PMS algorithm was specifically designed to deal with partially measurable systems and to take into account the presence of online and offline state estimators. The algorithm was shown to robustly learn from scratch how to control mechanical systems, in both simulated and real environments, facing correctly the velocity estimation problem. However, the tuning of accurate filters is a tedious task that could be particularly challenging and time-consuming for systems affected by significant noise.

In this chapter, instead, we present an alternative approach for learning to control mechanical systems, called *Derivative Free Monte Carlo Probabilistic Inference for Learning COntrol* (DF-MC-PILCO). This new approach circumvents the necessity of estimating velocities, by working only with the history of measured positions and control actions. We adopted a Derivative-Free (DF) predictive model, inspired by [77, 28], together with a control policy that depends only on past positions. DF-MC-PILCO entails the same three main phases of a model-based policy gradient approach depicted in Section 2.1.1, here reformulated in a DF sense.

We compare the proposed method with MC-PILCO4PMS on the same benchmarks studied in Chapter 3, namely, the simulated cart-pole, and the two real systems: the Furuta pendulum and the ball-and-plate. DF-MC-PILCO efficiently solved all the tasks with performance similar to MC-PILCO4PMS. Remarkably, DF-MC-PILCO managed to work directly with raw measurements, despite the presence of high noise. The observed results confirmed that the application of DF-MC-PILCO could be advantageous when working with particularly noisy mechanical systems.

The remainder of the chapter is structured as follows. In Section 4.1, we present the details of the proposed algorithm, DF-MC-PILCO, describing the characteristics of the DF model learning and policy update phases. Section 4.2 illustrates the studies conducted in the simulated cart-pole scenario to validate the algorithm. Section 4.3 reports the experiment performed on a simulated UR5 robot to test DF-MC-PILCO capacity of handling high DoF systems. Section 4.4 shows the results of the experiments with the two real mechanical systems. Finally, we draw conclusions in Section 4.5.

4.1 Derivative-Free MC-PILCO

Here we present the proposed DF-MC-PILCO algorithm, whose objective is to learn from scratch how to control mechanical systems without the need of performing any kind of velocity estimation. In fact, tuning effective estimators may be a tedious and problematic operation, especially in the presence of high measurement noise. DF-MC-PILCO circumvents these issues by adopting a DF formulation of the problem.

Recalling the notation introduced in the previous chapter, consider a mechanical system with $d_{\boldsymbol{q}}$ DoF, and denote with $\boldsymbol{q}_t \in \mathbb{R}^{d_{\boldsymbol{q}}}$ and $\dot{\boldsymbol{q}}_t \in \mathbb{R}^{d_{\boldsymbol{q}}}$, respectively, the joint positions and velocities. Let indicate with $\boldsymbol{u}_t \in \mathbb{R}^{d_{\boldsymbol{u}}}$ the vector of control actions.

We consider a DF model of the system dynamics, given in the following general form

$$\boldsymbol{q}_{t+1} = f_{\text{df}}(\boldsymbol{q}_t, \boldsymbol{q}_{t-1}, \dots, \boldsymbol{q}_{t-m_{\boldsymbol{q}}}, \boldsymbol{u}_t, \dots, \boldsymbol{u}_{t-m_{\boldsymbol{u}}}). \tag{4.1}$$

The joint positions at the next time step are predicted based on the history of the past positions, from t up to $t - m_{\boldsymbol{q}}$, and the history of applied control actions, from t up to $t - m_{\boldsymbol{u}}$. Let $m_{\boldsymbol{q}}$ and $m_{\boldsymbol{u}}$ be called, respectively, the position memory and the control memory of the DF model. In this new DF framework, it is convenient to redefine the state of the system as $\boldsymbol{x}_t = [\boldsymbol{q}_t^T, \dots, \boldsymbol{q}_{t-m_{\boldsymbol{q}}}^T, \boldsymbol{u}_{t-1}^T, \dots, \boldsymbol{u}_{t-m_{\boldsymbol{u}}}^T]^T$.

In the following, we present the details of the model learning and policy update phases of the DF-MC-PILCO algorithm, detailing how they have been adapted to the new DF formulation of the problem.

4.1.1 DF model learning

We employed the GPR framework of Section 2.1.2, but instead of considering a full state representation with velocities, we train a DF GP model of form (4.1). Let us denote with $q_t^{(i)}$ the position of the i-th joint at time t, and define $\Delta_{q_t}^{(i)} = q_{t+1}^{(i)} - q_t^{(i)}$, for $i \in \{1, \dots, d_{\boldsymbol{q}}\}$. The evolution of $\Delta_{q_t}^{(i)}$ for all i is modeled using a distinct GP, whose input depends upon $[\boldsymbol{q}_t^T, \dots, \boldsymbol{q}_{t-m_{\boldsymbol{q}}}^T, \boldsymbol{u}_t^T, \boldsymbol{u}_{t-1}^T, \dots, \boldsymbol{u}_{t-m_{\boldsymbol{u}}}^T]^T$. Trivially, the transition functions of the components $\boldsymbol{q}_{t-1}, \dots, \boldsymbol{q}_{t-m_{\boldsymbol{q}}}, \boldsymbol{u}_{t-1}, \dots, \boldsymbol{u}_{t-m_{\boldsymbol{u}}}$ are deterministic and known.

Experimentally, we found that it is beneficial in terms of data-efficiency (details follow in Section 4.2.3) to rearrange the GP input as

$$\tilde{\boldsymbol{x}}_t = \left[\boldsymbol{q}_t^T, \Delta_{\boldsymbol{q}_{t-1}}^T, \dots, \Delta_{\boldsymbol{q}_{t-m_{\boldsymbol{q}}}}^T, \boldsymbol{u}_t^T, \dots, \boldsymbol{u}_{t-m_{\boldsymbol{u}}}^T\right]^T, \tag{4.2}$$

where $\Delta_{\boldsymbol{q}_{t-i}} = (\boldsymbol{q}_{t-i+1} - \boldsymbol{q}_{t-i})$, for $i = 1, \dots, m_{\boldsymbol{q}}$, following the same notation used before when defining the GP targets.

In this way, we are providing the model with additional information about the rates of change observed inside the considered past position memory interval $m_{\boldsymbol{q}}$. Depending on the considered application, it may be convenient to further modify the GP input vector w.r.t. (4.2), in order to exploit particular characteristics of the considered quantities. For instance, we applied a *sin-cos* expansion to angular quantities during some of the experiments presented in the next sections.

4.1.2 DF particle-based policy gradient

The GP-based DF predictive model is then employed to optimize the policy parameters $\boldsymbol{\theta}$ following a particle-based policy gradient strategy, as described in Section 2.4. In fact, DF-MC-PILCO computes $\hat{J}(\boldsymbol{\theta})$, an approximation of the expected cumulative cost $J(\boldsymbol{\theta})$ (defined as in (2.1)), exploiting the posterior distribution $\boldsymbol{p}(\boldsymbol{x}_{t+1}|\tilde{\boldsymbol{x}}_t, \mathcal{D})$ given by the GPs. After that, it updates the parameters $\boldsymbol{\theta}$ through a gradient-based procedure.

Following the same strategy adopted by MC-PILCO, the computation of $\hat{J}(\boldsymbol{\theta})$ entails the simulation of the effects of the policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ on M independent state particles by cascading the one-step-ahead stochastic predictions. In particular, let $\boldsymbol{q}_t^{(m)}$, for $m = 1, \dots, M$, represent the position of the M state particles simulated by the DF GP model. Starting positions are sampled from a given distribution $p(\boldsymbol{q}_0)$. We assume that the system is not moving at $t = 0$, i.e., $\boldsymbol{q}_0^{(m)} = \boldsymbol{q}_{-1}^{(m)} = \cdots = \boldsymbol{q}_{-m_{\boldsymbol{q}}}^{(m)}$. At each time step t, in order to simulate the presence of measurement error, we corrupt the particle positions $\boldsymbol{q}_t^{(m)}$ with a fictitious noise $\boldsymbol{e}_t^{(m)}$, e.g. a zero mean Gaussian i.i.d. noise, obtaining a set of simulated measurements $\bar{\boldsymbol{q}}_t^{(m)} = \boldsymbol{q}_t^{(m)} + \boldsymbol{e}_t^{(m)}$. Then, for each particle, the

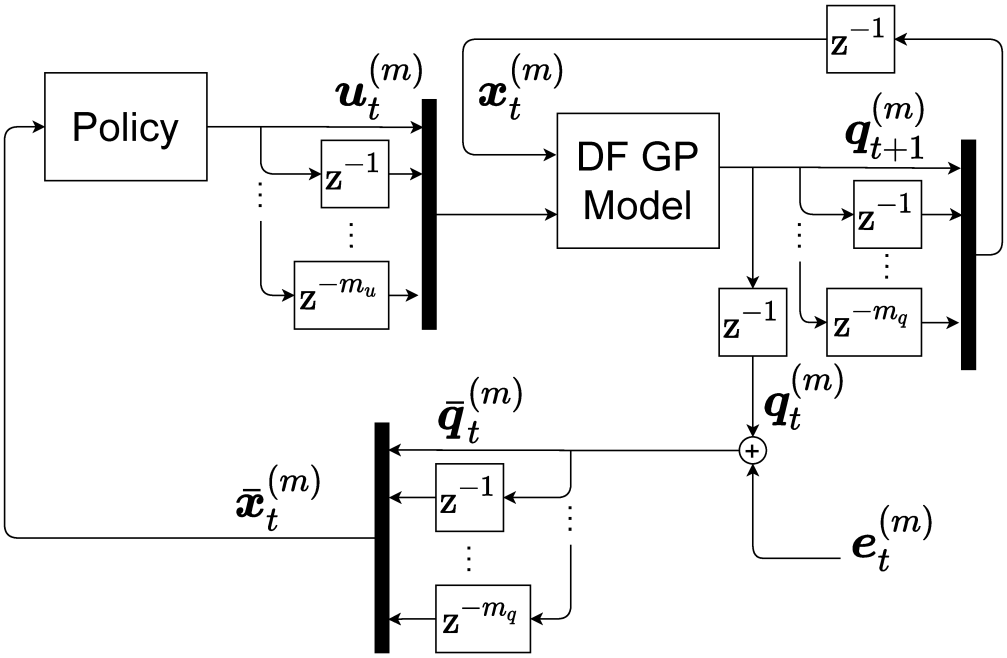


Figure 4.1: DF-MC-PILCO particles generation block schemes.

policy $\boldsymbol{\pi_\theta}$ selects the next control actions $\boldsymbol{u}_t^{(m)}$ according to the history of the simulated measurements, $\bar{\boldsymbol{x}}_t^{(m)} = [\bar{\boldsymbol{q}}_t^{(m)T}, \dots, \bar{\boldsymbol{q}}_{t-m_q}^{(m)T}]^T$. Finally, the M positions at the next time step, $t+1$, are simulated by forward sampling from the distributions derived by the DF GP model $\boldsymbol{p}(\boldsymbol{x}_{t+1}^{(m)}|\tilde{\boldsymbol{x}}_t^{(m)}, \mathcal{D})$ (for $m = 1 \dots M$) with $\tilde{\boldsymbol{x}}_t^{(m)}$ defined for each particle m as in (4.2). This procedure is iterated for T time steps, obtaining M different particle trajectories $\{\{\boldsymbol{x}_t^{(m)}\}_{m=1}^M\}_{t=0}^T$, that simulate the results of the policy. The particles generation procedure is depicted in the block scheme of Figure 4.1.

Analogously to MC-PILCO, the sample mean of the costs incurred by the different particles provides $\hat{J}(\boldsymbol{\theta})$, the estimate of the expected cumulative cost. Then, the resulting computational graph allows us to compute $\nabla_{\boldsymbol{\theta}}\hat{J}(\boldsymbol{\theta})$, i.e., the gradient of $\hat{J}(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$, by means of backpropagation, exploiting the reparametrization trick to propagate the gradient through the stochastic operations. Finally, a stochastic gradient descent algorithm can exploit the estimated gradient to update the policy parameters.

In this work, we considered again the *squashed-RBF-network* policy described in (2.10) with input vector given by

$$\boldsymbol{x}_t^* = \left[\boldsymbol{q}_t^T, \Delta_{\boldsymbol{q}_{t-1}}^T, \dots, \Delta_{\boldsymbol{q}_{t-m_q}}^T\right]^T, \tag{4.3}$$

where we are providing to the policy the same information about the differences between

consecutive position measures used for GP input in (4.2). Nevertheless, it is worth mentioning that we are not restricted to this particular policy structure, and DF-MC-PILCO can easily deal with any kind of differentiable policy.

4.2 Experimental Validation: Simulated Cart-pole

As a preliminary validation, we tested DF-MC-PILCO on the simulated cart-pole swing-up task to analyze its performance under different set-ups. We compare the proposed approach to MC-PILCO4PMS, the algorithm described in Chapter 3. The physical properties of the simulated system are the same that were provided in Section 2.5.

Recalling the previous notation, let p_t and α_t denote, respectively, the position of the cart and the the angle of the pole, at time step t, hence $\boldsymbol{q}_t = [p_t, \alpha_t]^T$. The cart-pole starts from its downward stable equilibrium at $\theta_0 = 0$ [rad] and $p_0 = 0$ [m]. The target configurations corresponding to the swing-up of the pendulum is still given by $p^{des} = 0$ [m] and $|\alpha^{des}| = \pi$ [rad]. Hence, the cost function we aim to minimize is the same considered in the previous chapters and defined in (2.16). The control action is the force that pushes the cart horizontally and the system is controlled at 30 [Hz]. We considered a Gaussian measurement noise with standard deviation of 10^{-3} [m] for cart positions and $2 \cdot 10^{-3}$ [rad] for pole angles.

The GPs of the DF model are equipped with the SE kernel described in (2.2). The policy adopted is a *squashed-RBF-network policy* with $n_b = 200$ basis functions and $u_{max} = 10$ [N]. The number of particles is set to $M = 400$ during policy optimization. In order to avoid singularities due to the angles, we replaced, in both the model inputs $\tilde{\boldsymbol{x}}_t$ defined in (4.2) and policy input $\boldsymbol{x}^*$ defined in (4.3), occurrences of α_t with $sin(\alpha_t)$ and $cos(\alpha_t)$. Exploration data were collected by random actions, obtained by filtering Gaussian white noise with cut-off frequency 1.5 [Hz].

The objective is to analyze different DF-MC-PILCO setups and compare their performance with the results obtained by MC-PILCO4PMS, as a benchmark. We analyzed the results obtained in 50 distinct experiments, consisting of 5 trials of length 3 seconds, varying the random seed each time. In this way, it is possible to evaluate the robustness of the algorithm to different exploration trajectories and policy initialization, as well as to different noise realizations. In particular, we investigate the effects that different position and control memories, $m_{\boldsymbol{q}}$ and $m_{\boldsymbol{u}}$, have on modeling and policy learning.

We studied four different DF-MC-PILCO setups, choosing the value of $m_{\boldsymbol{q}}$ between 1 and 2, and $m_{\boldsymbol{u}}$ between 0 and 1. In the following, we will refer to these different alternatives with the symbol $DF^{m_{\boldsymbol{u}}}_{m_{\boldsymbol{q}}}$.

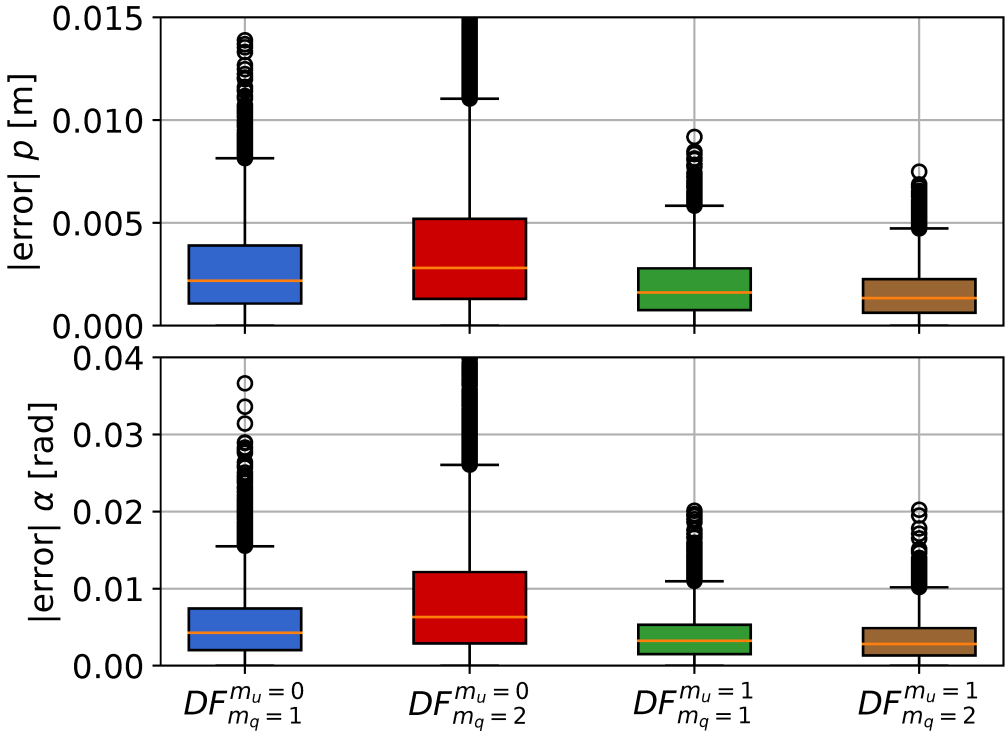


Figure 4.2: Absolute values of prediction errors, for both p and α, obtained by different DF GP models at trial 5 in the simulated cart-pole experiments.

4.2.1 Modeling results

We compared the accuracy of the different DF GP models by looking at the absolute values of the prediction errors observed on the data registered at the last trial in all the 50 experiments. Models were trained using all the data collected up to that trial. Figure 4.2 reports the results by means of box-plots, showing median values, confidence intervals, and outliers. The results show that the presence of input history, $\boldsymbol{u}_{t-1}$, as part of the GP inputs is beneficial, and within this choice, see models with $m_{\boldsymbol{u}} = 1$, the best results are obtained by $DF^{m_{\boldsymbol{u}}=1}_{m_{\boldsymbol{q}}=2}$. On the other hand, the greater errors and the significant number of outliers obtained by models with $m_{\boldsymbol{u}} = 0$ seem to indicate that these kinds of setups are not fully capable of fitting the registered position changes. Also, it appears that using a longer position memory leads to an improvement of prediction accuracy only when the control memory is $m_{\boldsymbol{u}} = 1$. We can conclude that, in the setup considered, it seems beneficial to provide DF GP models with information about past control actions ($m_{\boldsymbol{u}} = 1$) for fitting the dynamics of the system without relying on velocity estimates.

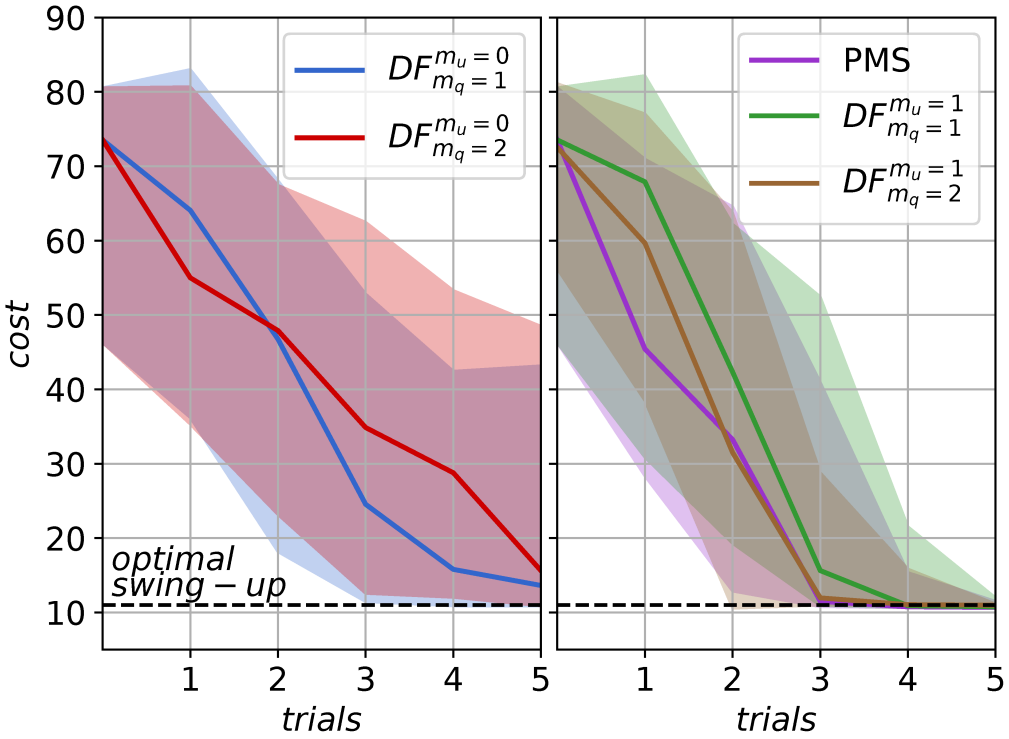


Figure 4.3: Cumulative costs registered during simulated cart-pole experiments from the four considered DF-MC-PILCO setups and the MC-PILCO4PMS (indicated by the shorthand PMS). The cost associated with an optimal swing-up is reported as a reference value.

4.2.2 Policy learning results

In this section, we evaluate the performance of the control policies learned by the different DF-MC-PILCO setups and by MC-PILCO4PMS. The cumulative costs and success rates obtained at each trial in the 50 experiments are reported, respectively, in Figure 4.3 and in Table 4.1. In the two plots the cumulative cost is reported in terms of median values and confidence intervals defined by the 5-th and 95-th percentiles.

As one would expect, the worse modeling results of $DF^{m_u=0}_{m_q=1}$ and $DF^{m_u=0}_{m_q=2}$ lead to an unsatisfactory policy learning. These DF-MC-PILCO setups manage to complete a successful a swing-up only in, approximately, half of the cases. On the other hand, when using $m_u = 1$, DF-MC-PILCO is able to robustly find an optimal solution for the task by trial 5. In particular, the performance of $DF^{m_u=1}_{m_q=2}$ are almost equivalent to the results of MC-PILCO4PMS. This result confirms the effectiveness of the proposed method: with less information, as we are not manually tuning any velocity estimator, DF-MC-PILCO achieves state-of-the-art performance. For the user, this corresponds to less effort and a more general method without compromising significantly the performance.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
$DF^{m_{\boldsymbol{u}}=0}_{m_{\boldsymbol{q}}=1}$	0%	14%	34%	46%	56%
$DF^{m_{\boldsymbol{u}}=0}_{m_{\boldsymbol{q}}=2}$	0%	10%	18%	28%	52%
$DF^{m_{\boldsymbol{u}}=1}_{m_{\boldsymbol{q}}=1}$	0%	8%	52%	86%	96%
$DF^{m_{\boldsymbol{u}}=1}_{m_{\boldsymbol{q}}=2}$	0%	20%	73%	93%	100%
PMS	0%	14%	82%	98%	96%

Table 4.1: Success rates obtained by the four different DF-MC-PILCO setups and by MC-PILCO4PMS (indicated with the shorthand PMS) at each trial of the simulated cart-pole experiments.

4.2.3 Analysis of input vector structure

Before concluding this section, we would like to analyze the reasons behind the decision to use (4.2) and (4.3) as GP and policy input vectors, respectively. In this respect, we compared the results obtained by DF-MC-PILCO with position memory $m_{\boldsymbol{q}} = 2$ and control memory $m_{\boldsymbol{u}} = 1$ using two different structures for the input vectors. The first employs directly the history of positions and actions up to time step t as GP input, e.g., $\tilde{\boldsymbol{x}}_t = [\boldsymbol{q}_t^T, \dots, \boldsymbol{q}_{t-m_{\boldsymbol{q}}}^T, \boldsymbol{u}_t^T, \dots, \boldsymbol{u}_{t-m_{\boldsymbol{u}}}^T]^T$, and the history of positions as policy input, e.g., $\boldsymbol{x}_t^* = [\boldsymbol{q}_t^T, \dots, \boldsymbol{q}_{t-m_{\boldsymbol{q}}}^T]^T$. The second version is the one employed previously with GP input and policy input defined as in (4.2) and (4.3), respectively. To distinct the two implementations, we labeled the first as $DF^{m_{\boldsymbol{u}}=1}_{m_{\boldsymbol{q}}=2}$ *naive*, and the second $DF^{m_{\boldsymbol{u}}=1}_{m_{\boldsymbol{q}}=2}$ *with position differences*. We analyzed the results obtained by these two setups in 50 distinct experiments, consisting of 7 trials of length 3 seconds, varying the random seed each time. The obtained cumulative costs are reported in Figure 4.4 in terms of median values and confidence intervals defined by the 5-th and 95-th percentiles.

It is clear that providing information about the rate of change of position measures, by using (4.2) and (4.3) as input vectors, greatly improves the data efficiency of DF-MC-PILCO algorithm. In fact, the $DF^{m_{\boldsymbol{u}}=1}_{m_{\boldsymbol{q}}=2}$ *naive* implementation (that uses directly the history of position and controls) shows a much slower convergence, reaching a 79% success rate only at trial 7. On the other hand, we have already seen, that $DF^{m_{\boldsymbol{u}}=1}_{m_{\boldsymbol{q}}=2}$ *with position differences* is able to always find a solution by trial 5.

This result underlines the importance of the information carried out by the differences between consecutive measured positions. Without this information, the model needs more data to correctly capture the dynamics of the system relying only on positions. Through input vectors (4.2) and (4.3), we are able to provide the model with knowledge about a sort of velocity, without requiring any kind of filtering procedure.

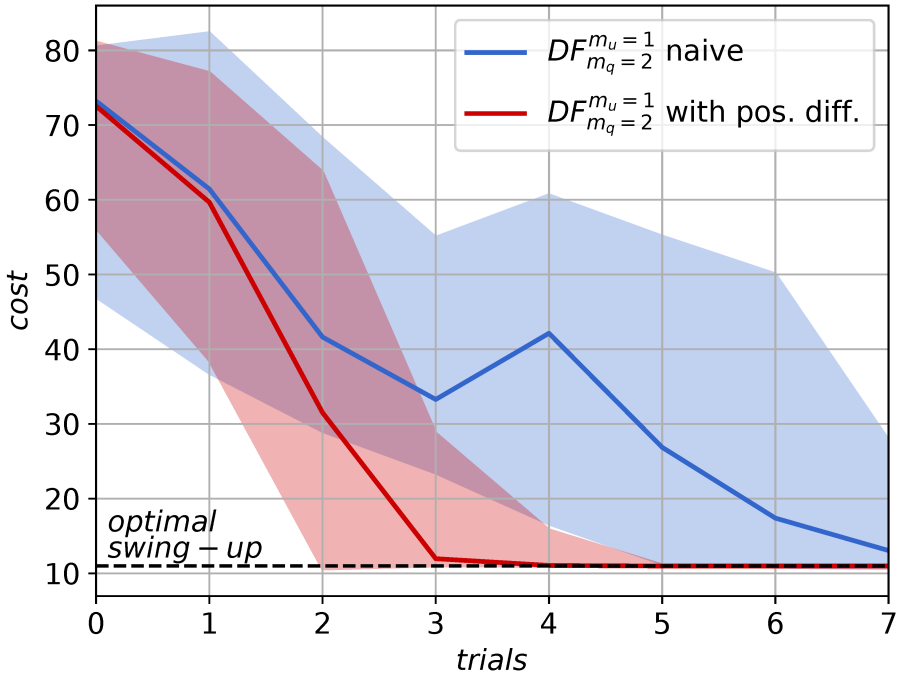


Figure 4.4: Cumulative costs registered by DF-MC-PILCO varying the structures of the GP and policy input vectors. The cost associated with an optimal swing-up is reported as a reference value.

4.3 High DoF Experiment: Simulated UR5 Robot

The objective of this experiment is to test DF-MC-PILCO in a more complex system with higher DoF. We used DF-MC-PILCO to learn a joint-space torque controller for a UR5, a robotic manipulator with 6 DoF, simulated in MuJoCo [93], assuming to measure only joint angles and not velocities. Measurements are perturbed by the presence of a white Gaussian noise with standard deviation of 10^{-3}. Let us denote with $\boldsymbol{q}_t \in \mathbb{R}^6$ the joint angles and with $\boldsymbol{u}_t \in \mathbb{R}^6$ the applied torques. Our objective is to learn a DF control policy able to follow a desired trajectory $\{\boldsymbol{q}^r_t\}^T_{t=1}$. Let $\boldsymbol{e}_t = \boldsymbol{q}^r_t - \boldsymbol{q}_t$ denote the position error at time t. DF-MC-PILCO memories were set to $m_{\boldsymbol{q}} = 2$ and $m_{\boldsymbol{u}} = 1$, hence the DF state of the system at time step t is defined as $\boldsymbol{x}_t = [\boldsymbol{q}^T_t, \boldsymbol{q}^T_{t-1}, \boldsymbol{q}^T_{t-2}, \boldsymbol{u}^T_{t-1}]^T$. The GP input vector was defined applying a *sin-cos* expansion of angular quantities as $\tilde{\boldsymbol{x}}_t = [sin(\boldsymbol{q}_t)^T, cos(\boldsymbol{q}_t)^T, \Delta^T_{\boldsymbol{q}_{t-1}}, \Delta^T_{\boldsymbol{q}_{t-2}}, \boldsymbol{u}^T_t, \boldsymbol{u}^T_{t-1}]^T$.

The policy adopted was a multi-output *squashed-RBF-network* with $n_b = 400$ basis functions and $u_{max} = 1$ [N·m] for all the joints. $M = 200$ particles were used during optimization. The policy takes in input the vector $\boldsymbol{x}^*_t = [sin(\boldsymbol{q}_t)^T, cos(\boldsymbol{q}_t)^T, \Delta^T_{\boldsymbol{q}_{t-1}}, \Delta^T_{\boldsymbol{q}_{t-2}}, \boldsymbol{e}^T_t]^T$. Figure 4.5 represents the overall control scheme.

In this experiment, we considered a control horizon of 4 seconds with a sampling

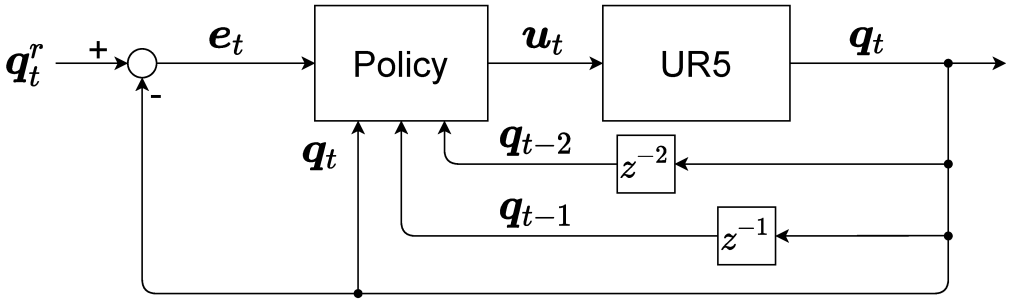
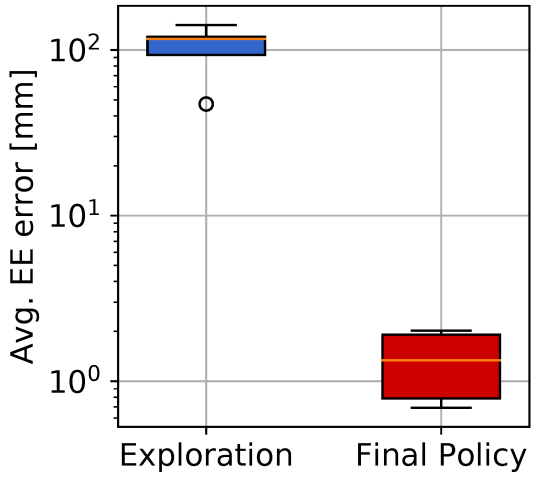


Figure 4.5: DF-MC-PILCO control scheme in the simulated UR5 experiment.

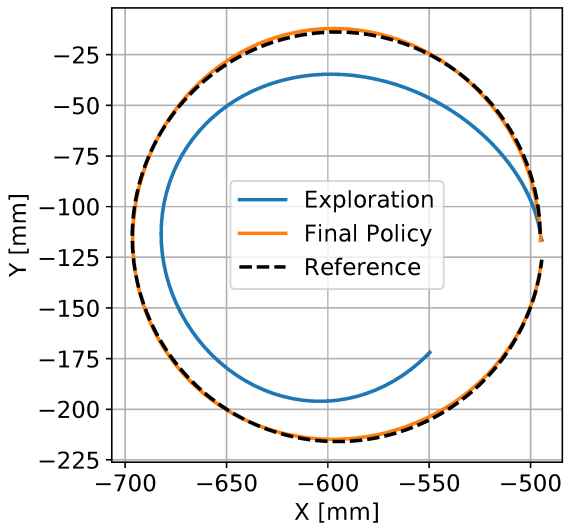
time of 0.02 seconds. The reference trajectory has been calculated to make the end-effector to draw a circle in the X-Y operational space. The initial exploration, used to initialize the DF GP model, is provided by a poorly-tuned PD controller (for which we estimated velocity by backward differentiation). We used $M = 200$ of particles for gradient estimation, and considered the following cost function,

$$c(\boldsymbol{x}_t) = 1 - \exp\left(||\boldsymbol{q}_t^r - \boldsymbol{q}_t||^2\right).$$

The experiment was repeated 10 different times, varying the random seed and the initial exploration trajectories, obtained each time by using random PD gains, uniformly sampled from $K_P \sim \mathcal{U}(0.5, 2)$ and $K_D \sim \mathcal{U}(0.01, 0.2)$.



(a) Average end-effector position tracking errors obtained during the exploratory phases and by the DF-MC-PILCO control policy learned at trial 3.



(b) Example of end-effector trajectories obtained during exploration and by the trained policy.

Figure 4.6: Experimental results on simulated UR5 robot.

DF-MC-PILCO managed to learn an effective control policy by the third trial in all the repetitions. The average end-effector tracking errors obtained are reported in Fig. 4.6a, where results are given by means of box plots. Indicatively, we observed average positioning errors not superior to 2 [mm]. Finally, in Fig. 4.6b we reported, as an example, the exploratory and final trajectories obtained during one of the test. We can consider the obtained performance satisfying, also w.r.t. the results obtained in Chapter 2, where an effective controller was learned one trial earlier, but relying on direct velocity measurements, which is an unrealistic assumption, possible only in simulation.

4.4 Experiments with Real Systems

In this section, we report the results obtained by DF-MC-PILCO when applied to real systems. In particular, we experimented on the same two benchmarks previously described in Section 3.4. The objective is to compare the performance obtained by DF-MC-PILCO in these two setups with the results of MC-PILCO4PMS reported in the previous chapter.

4.4.1 Furuta pendulum

The Furuta pendulum considered here is the same described in Section 3.4.1. We denote the pose at time step t with $\boldsymbol{q}_t = [\alpha_t^h, \alpha_t^v]^T$, where α_t^h is the angle of the horizontal joint and α_t^v the angle of the vertical joint attached to the pendulum. The objective is to learn a DF controller able to swing-up the pendulum and stabilize it in the upward equilibrium ($\alpha_t^v = \pm\pi$ [rad]) with $\alpha_t^h = 0$ [rad], starting from $\boldsymbol{q}_0 = [0, 0]^T$. The trial length is 3 [s] and the system is controlled at a sampling frequency of 30 [Hz]. The cost function adopted is the same considered in the MC-PILCO4PMS experiment, that was defined in (3.2).

In this scenario, we used position memory $m_{\boldsymbol{q}} = 2$ and control memory $m_{\boldsymbol{u}} = 1$. We equipped the DF GP model with an SE kernel and adopted a *squashed-RBF-network* with $n_b = 200$ basis functions as control policy. $M = 400$ particles were simulated during policy optimization. We replaced, in both GP inputs $\tilde{\boldsymbol{x}}_t$ and policy input $\boldsymbol{x}^*$, occurrences of α_t^h and α_t^v with their *sin-cos* expansion, as previously done in the simulated cart-pole case. The exploration trajectory has been obtained using as input a sum of ten cosine waves of random frequencies and same amplitudes. The presence of quantization errors was simulated during particles generation by corrupting predicted angles with a uniform fictitious measurement noise $\mathcal{U}(\frac{-\pi}{4096}, \frac{\pi}{4096})$ [rad].

DF-MC-PILCO learned how to swing-up the Furuta pendulum at trial 6, i.e. after 18 seconds of experience. That is the same result obtained by MC-PILCO4PMS, when using the SE kernel. Hence, the DF approach showed no particular differences in terms

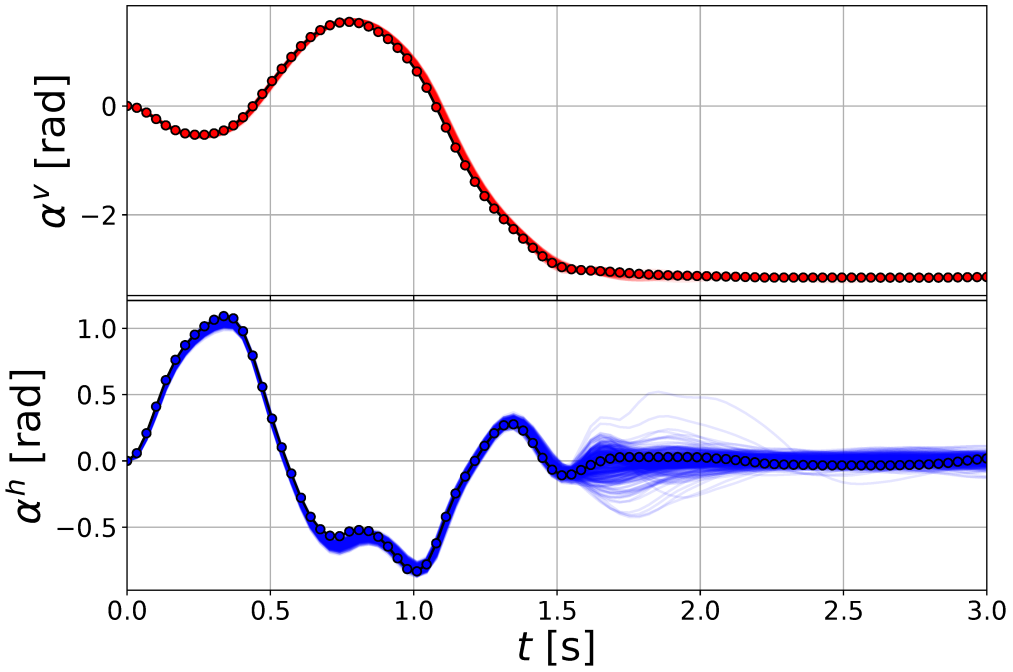


Figure 4.7: Real swing-up trajectory (bullets) and particles prediction (shaded lines) obtained by DF-MC-PILCO at trial 6 of the Furuta pendulum experiment.

of data-efficiency when compared with an approach that makes use of velocity estimates. In Figure 4.7, we report the successful swing-up performed by DF-MC-PILCO at trial 6, together with the particles predicted by the DF GP model, simulating the effects of the same control policy. Notice how the particles' trajectories resemble almost perfectly the real behaviour of the two angles.

4.4.2 Ball-and-plate

Furthermore, we applied DF-MC-PILCO one the same ball-and-plate system that was previously introduced in Section 3.4.2. We remind the reader that, at each time step t, b_t^x and b_t^y indicate, respectively, the X and Y coordinates of the center of the ball on the plate, while, $alpha_t^{(1)}$ and $\alpha_t^{(2)}$ are the angles of the two motors tilting the plate. Thus, it is possible to define $\boldsymbol{q}_t = [b_t^x, b_t^y, \alpha_t^{(1)}, \alpha_t^{(2)},]^T$.

As mentioned in the previous chapter, the drivers of the motors allow only position control, and do not provide feedback about the motors' angles. To keep track of the motor angles, we defined the control actions as the difference between two consecutive reference values sent to the motor controllers, limiting the maximum input to a sufficiently small value, i.e. 4 [deg], such that the motor controllers are able to reach the target angle within the sampling time. Then, we can confidently assume that the reference angles

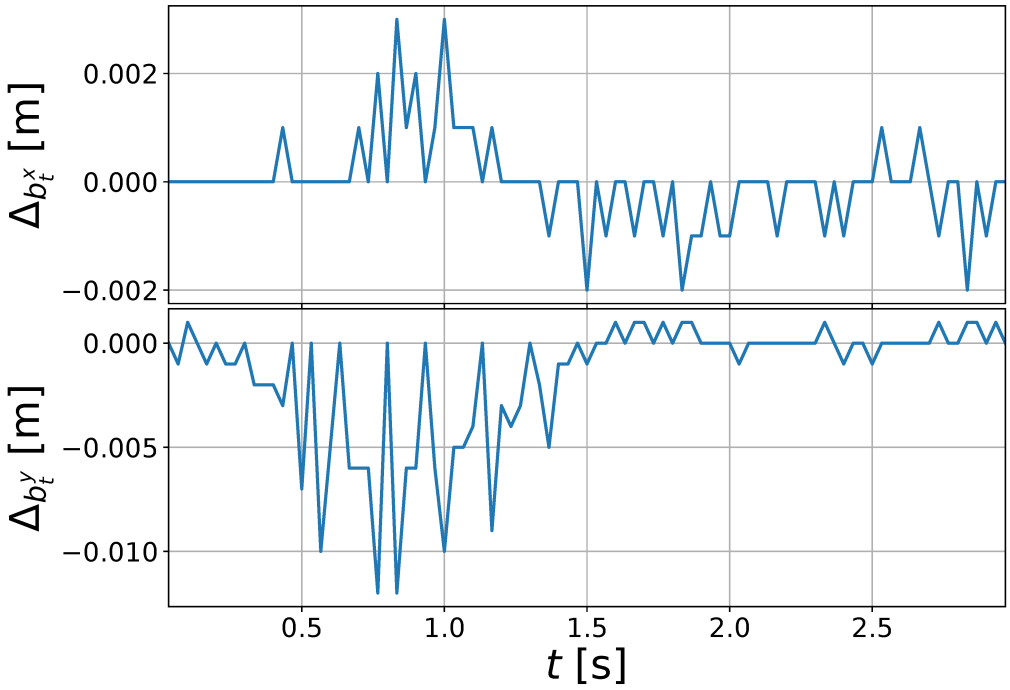


Figure 4.8: Example of GP targets in the ball-and-plate experiment, i.e. measured ball position changes in X and Y directions.

and the motor angles coincide, and we have $u_t^{(1)} = \alpha_{t+1}^{(1)} - \alpha_t^{(1)}$ and $u_t^{(2)} = \alpha_{t+1}^{(2)} - \alpha_t^{(2)}$. The objective of the experiment is to learn how to control the motor angles in order to stabilize the ball around the center of the plate. The trial length is 3 seconds, with a sampling frequency of 30 [Hz]. The cost function encoding the task is the same used by MC-PILCO4PMS and it is given by (3.3).

With regards to the DF model setup, we considered position memory $m_{\boldsymbol{q}} = 2$ and control memory $m_{\boldsymbol{u}} = 1$, and we replaced in both GP inputs $\tilde{\boldsymbol{x}}_t$ and policy input $\boldsymbol{x}^*$, the occurrences of $\alpha_t^{(1)}$ and $\alpha_t^{(2)}$ with their *sin-cos* expansion. Analogously to the previous MC-PILCO4PMS experiment, the kernel function of the DF GP model is given by the sum of a SE kernel that takes as input the whole GP input vector, and of a linear kernel that takes as input only the *sin-cos* expansion of angular quantities. The control policy is a *squashed-rbf-network* with $n_b = 400$ basis functions. Policy optimization involves the use of $M = 400$ particles.

The initial exploration is implemented in two trials, in which the control signals are two distinct noisy triangular waves. Mostly during exploration and initial trials, the ball might touch the borders of the plate. In those cases, we kept data up to the collision instant, and discarded it thereafter. The presence of quantization errors was simulated during particles generation by corrupting predicted angles with a uniform fictitious

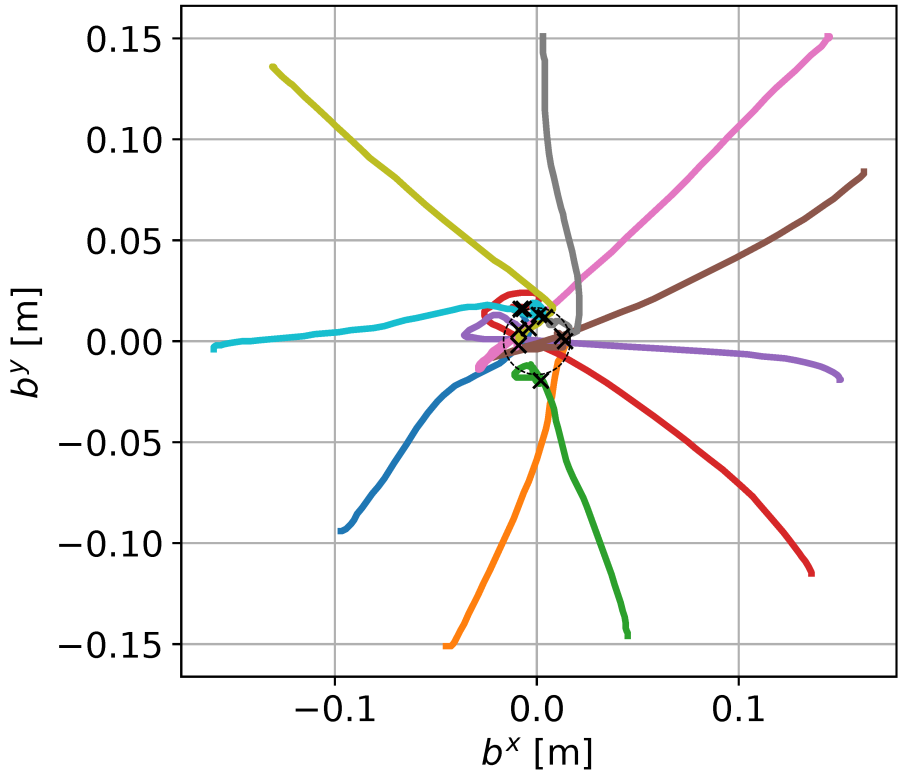


Figure 4.9: Ten different ball trajectories obtained by DF-MC-PILCO policy. Steady-state positions are marked with black crosses. The dashed circle has the same diameter as that of the ball.

measurement noise $\mathcal{U}(\frac{-0.001}{2}, \frac{0.001}{2})$ [m]. A peculiarity of this experiment in comparison to the others seen before is a wide range of initial conditions. In fact, the policy must learn how to control the ball to the center starting from any position on the plate's surface. Hence, the initial distribution considered for b_0^x and b_0^y is the uniform $\mathcal{U}(-0.15, 0.15)$ [m]. The measurements provided by the camera are affected by a significant quantization error. For instance, please refer to Figure 4.8, where we reported the measured differences between consecutive ball positions during a trial. Consider that these quantities are the targets of the GPs in the DF model. In such a context, the estimation of the ball velocity can be very challenging. In fact, for applying MC-PILCO4PMS on the same system, the tuning of online and offline filters was a delicate and time-consuming procedure of critical importance for the success of the algorithm. On the contrary, DF-MC-PILCO managed to solve the task working directly with raw position measurements, without the need of applying any kind of filtering. Besides that, DF-MC-PILCO proved to be surprisingly data-efficient, being able to learn a successful policy at the second trial, after only 7.97 seconds of experience, whereas MC-PILCO4PMS solved the task after 11.33

seconds. In Figure 4.10, we reported the trajectories observed at each trials.

We tested the policy starting from ten different points in order to compare the two policies obtained by DF-MC-PILCO (Figure 4.9) and MC-PILCO4PMS (see Figure 3.7, in the previous chapter). The mean steady-state error, i.e. the average distance of the last ball position from the center observed in the ten tests, was 0.0134 [m], while MC-PILCO4PMS final policy obtained a slightly better result, with a mean error of 0.0099 [m]. Nevertheless, this difference in terms of performance is quite negligible, given the dimension of the ball whose radius is 0.016 [m], and the the fact that DF-MC-PILCO worked directly on raw measurements.

4.5 Conclusion

We presented DF-MC-PILCO, an extension of the MC-PILCO algorithm of Chapter 2, specifically designed to learn from scratch how to control mechanical systems without the need of computing any explicit velocity estimates. In our opinion, this may be a critical advantage when dealing with real mechanical systems affected by significant measurement noise, since, in this kind of scenario, the design of accurate velocity estimators can be a challenging and time-consuming procedure. The algorithm uses GPR to model the joint position changes, based upon the history of past control actions and measurements. DF-MC-PILCO was tested both in simulated environments (cart-pole and UR5 robot) as well as in two different real systems (Furuta pendulum and ball-and-plate). It proved able to solve all the tasks, with results that are in line with the performance of MC-PILCO4PMS algorithm, that works with a complete state representation and must estimate velocities.

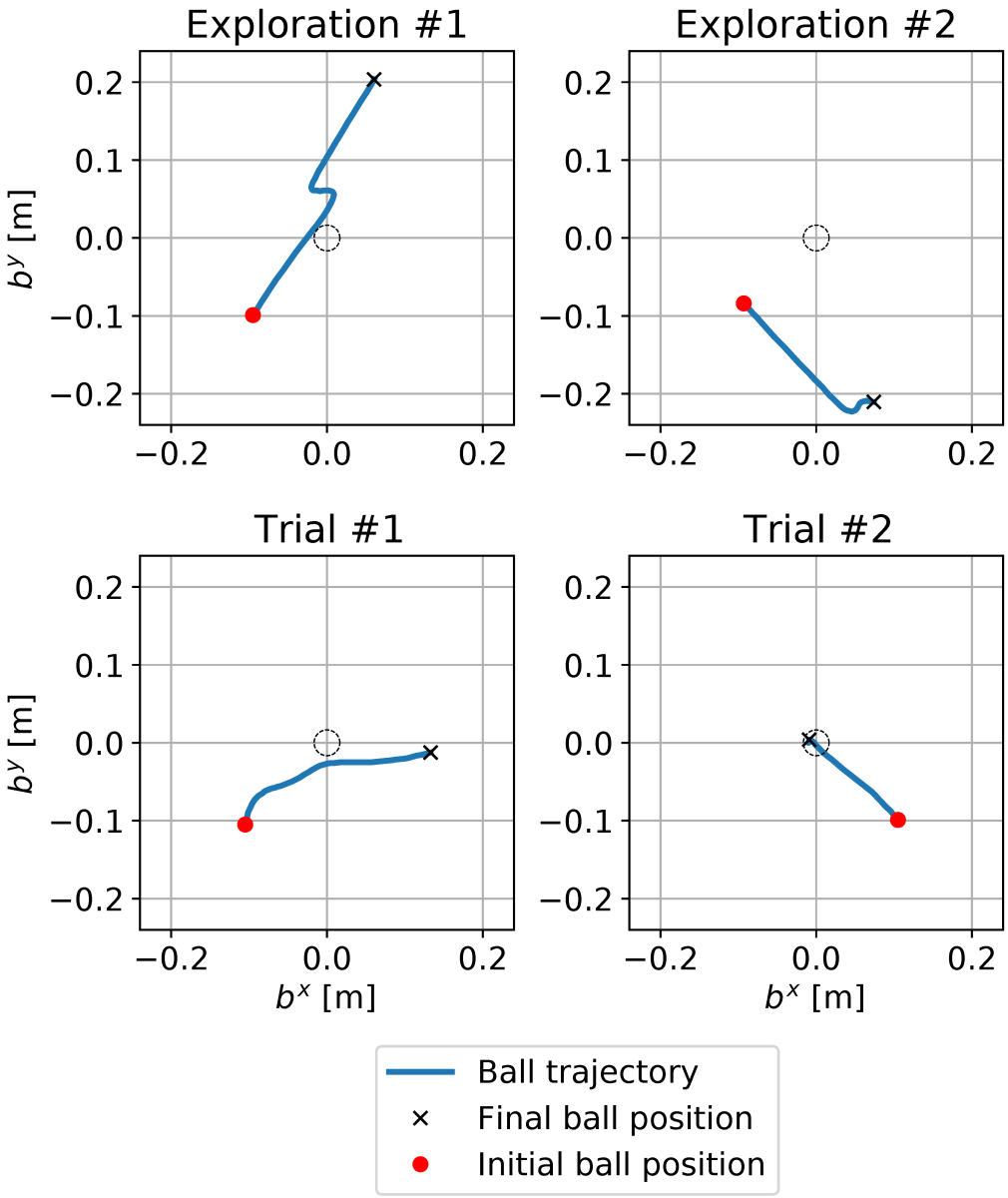


Figure 4.10: Ball trajectories observed during initial explorations and the trials attempted by DF-MC-PILCO. Initial and final ball positions are indicated as a red circle and a black cross, respectively. The dashed circles are centered in the target positions and have the same diameter of the used ball.

Part II

Gaussian Processes for Data-Driven Robot Control

5

Feedback Linearization Torque Control using Gaussian Processes

Dynamics models are fundamental in robotics. For instance, inverse dynamics models, which relate joint trajectories to joint torques, are used in high-precision trajectory tracking applications [91, 32, 86], and also in problems where robots interact with the environment, such as force control [87, 86], impedance control [44, 16], and collision detection [43, 29].

In the aforementioned applications, the accuracy of the inverse dynamics model is crucial. However, deriving an accurate model of the robot inverse dynamics is a challenging task, in particular when system specifications are limited or uncertain, or when complex behaviors such as friction and elasticity are relevant. Indeed, in these contexts, the identification of parametric models derived from first principles of physics [88, 85] are often not effective, due to model bias and unmodeled behaviors. For these reasons, in the last decades, several black-box and grey-box strategies for inverse dynamics identification have been proposed. A relevant class of solutions is based on Gaussian Process Regression (GPR) [74], see for instance [60, 68, 79, 84, 75]. Here, instead of identifying the physical parameters of the model, the inverse dynamics are treated as an unknown function, which relates position, velocity, and acceleration of the joints to torques. This unknown function is modeled a priori as a Gaussian Process (GP), with

covariance parametrized through a kernel function [74, 83]. The posterior distribution of the joint torques, given the observed data, can be computed in closed form, and can be used to predict joint torques.

Compared to physical models, GP models are less interpretable since they are not strictly related to the dynamics equations; consequently, their use in control applications might be less straightforward. However, several works show that such models can be used in applications, see, for instance, [84, 69, 2, 8, 75] concerning trajectory tracking, and [29] concerning proprioceptive collision detection. Typically, in trajectory tracking, GP models are exploited by implementing a feedforward control scheme [24], see the diagram in Figure 5.1a. Instead of using parametric models, in the GP implementation, the feedforward term is the output of the GP model evaluated for the position, velocity, and acceleration of the reference trajectory. The control loop is closed with a decentralized PD controller to cancel errors. When the GP model is accurate and the PD gains are set properly, the feedback loop is effective in canceling the residual tracking error. However, there are some issues that could limit the performance of a feedforward controller.

(i) In the feedback loop, coupling between different degrees of freedom (DoF) are not considered.

(ii) The robot inertia is configuration dependent, and, in some cases, it might be difficult to obtain a set of PD gains that can assure the same performance for all configurations.

(iii) Convergence is not guaranteed, even if the inverse dynamics is exactly known.

For details about (ii) and (iii), we refer the interested reader to Chapter 10.5 of [24].

An alternative control scheme is feedback linearization control [91, 32, 86], described in the diagram in Figure 5.1b. In contrast to feedforward control, where the model is used to compute a proper control input in advance, in feedback linearization, the inverse dynamics model is used to obtain a tracking error with linear dynamics. The control input is the sum of two terms. The first aims at compensating all the torques independent of accelerations. The second is given by a feedforward term proportional to the reference acceleration, and a PD feedback term. To account for couplings and variations of the inertia matrix, the second term is computed using an estimate of the inertia matrix. In contrast to feedforward control, feedback linearization assures asymptotic convergence, if the dynamics are known exactly. Moreover, the error dynamics are described by a second-order linear differential equation, fully characterized by the PD gains, providing a principled way to set the PD gains [86].

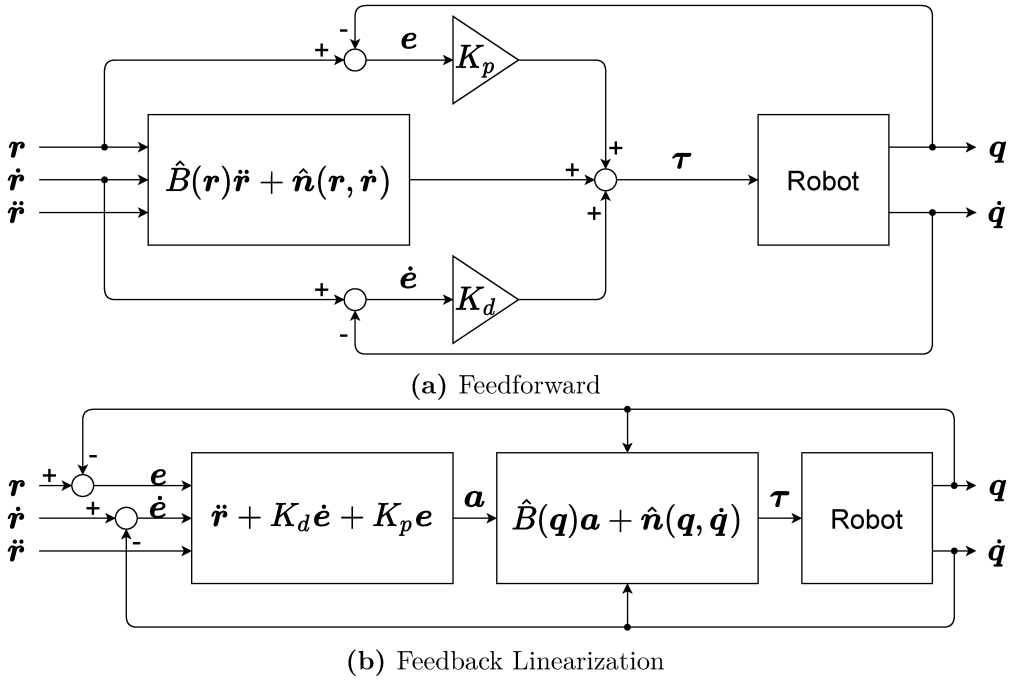


Figure 5.1: Diagrams of feedforward and feedback linearization control. $\boldsymbol{q}$, $\boldsymbol{r}$, and $\boldsymbol{e}$ are the joints position, the reference trajectory, and the tracking error.

In this work, we analyze two implementations of feedback linearization control based on GP models. The first implementation is simpler, and estimates directly the feedback linearization control input using the GP model. In contrast, the second implementation is composed of two steps. First, the inertia matrix and the compensation of the torques independent of accelerations are estimated separately by means of the GP model. Then, the feedback linearization control input is computed by applying its standard form. To the best of our knowledge, the first implementation has been attempted before only in [69]. However, that article was focused on issues related to modeling, and not to control. In contrast, the second implementation has never been proposed before, and it requires the estimation of several different components of the dynamics equations from the GP model, which is introduced in this work. We tested the two implementations with a simulated 7-DoF manipulator, varying also the choice of the GP prior, i.e., its kernel. The obtained results show that the second implementation is more robust w.r.t. the kernel choice and initial errors.

The remainder of this chapter is organized as follows. In Section 5.1, we provide background formulations of robot dynamics and control, as well as GPR. Section 5.2 describes the strategy proposed to estimate several different dynamics components from black-box GP models, and in Section 5.3 we describe the two feedback linearization

algorithms implemented. Experiments are described in Section 5.4, and conclusions are drawn in Section 5.5.

5.1 Background

In the first part of this section, we provide background formulation of robot dynamics, as well as introduce the trajectory tracking problem and describe feedback linearization control. In the second part, we describe GPR for inverse dynamics identification, detailing the black-box priors adopted in this work.

5.1.1 Robot dynamics and control

Consider a mechanical systems with n DoF, and denote with $\boldsymbol{q}_t \in \mathbb{R}^n$ its generalized coordinates at time t; $\dot{\boldsymbol{q}}_t$ and $\ddot{\boldsymbol{q}}_t$ are the velocity and the acceleration of the joints, respectively. The generalized torques, i.e., the control input of the system, are denoted with $\boldsymbol{\tau}_t \in \mathbb{R}^n$. For compactness, in the following, we will denote explicitly the dependencies on t only when strictly necessary. Under rigid body assumptions, the dynamics equations of a mechanical system are described by the following matrix equation

$$B(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{F}(\dot{\boldsymbol{q}}) = \boldsymbol{\tau}, \tag{5.1}$$

where $B(\boldsymbol{q})$ is the inertia matrix, while $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, $\boldsymbol{g}(\boldsymbol{q})$, and $\boldsymbol{F}(\dot{\boldsymbol{q}})$ account, respectively, for the contributions of fictitious forces, gravity, and friction, see [86] for a more detailed description. For compactness, we introduce also $\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{F}(\dot{\boldsymbol{q}})$. In the following, we will denote with $\hat{B}(\boldsymbol{q})$ and $\hat{\boldsymbol{n}}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ the estimates of $B(\boldsymbol{q})$ and $\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})$.

The trajectory tracking problem consists in designing a controller able to follow a reference trajectory $\boldsymbol{r}_t, \dot{\boldsymbol{r}}_t, \ddot{\boldsymbol{r}}_t$, starting from initial conditions $\boldsymbol{q}_{t_0}, \dot{\boldsymbol{q}}_{t_0}, \ddot{\boldsymbol{q}}_{t_0}$.

In feedback linearization control, the control input $\boldsymbol{\tau}$ is

$$\boldsymbol{a} = \ddot{\boldsymbol{r}} + K_p \boldsymbol{e} + K_d \dot{\boldsymbol{e}}, \tag{5.2a}$$

$$\boldsymbol{\tau} = \hat{B}(\boldsymbol{q})\boldsymbol{a} + \hat{\boldsymbol{n}}(\boldsymbol{q}, \dot{\boldsymbol{q}}). \tag{5.2b}$$

Assuming that the model is known exactly, i.e., $\hat{B}(\boldsymbol{q}) = B(\boldsymbol{q})$ and $\hat{\boldsymbol{n}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, combining (5.1) and (5.2) and recalling that $B(\boldsymbol{q})$ is invertible, it can be proven that the tracking error goes asymptotically to zero if $K_p > 0$ and $K_d > 0$ [86]. Indeed, under these assumptions, the dynamics of the tracking error is described by the following second

order linear differential equation

$$\ddot{\boldsymbol{e}} + K_d\dot{\boldsymbol{e}} + K_p\boldsymbol{e} = 0, \tag{5.3}$$

which is stable if $K_p > 0$ and $K_d > 0$. This fact represent a considerable advantage w.r.t. feedforward control, since it provides a principled way to chose K_p and K_d. Indeed, selecting $K_p = \omega^2 I$ and $K_d = 2\zeta\omega I$, with I being the identity matrix, we obtain n decoupled second-order input/output relations with natural frequency ω and damping ratio ζ.

5.1.2 GPR for inverse dynamics identification

GPR provides a solid probabilistic framework to identify the inverse dynamics from data. Typically, in GPR, each joint torque is modeled by a distinct and independent GP. Consider an input/output dataset $\mathcal{D} = \left\{\boldsymbol{y}^{(i)}, X\right\}$, where $\boldsymbol{y}^{(i)} \in \mathbb{R}^N$ is a vector collecting N measurements of $\tau^{(i)}$, the i-th joint torque, while $X = \{\boldsymbol{x}_{t_1} \dots \boldsymbol{x}_{t_N}\}$; $\boldsymbol{x}_t$ is the vector collecting the position, velocity and acceleration of the joints at time t, hereafter denoted GP input. The probabilistic model of $\mathcal{D}$ is

$$\boldsymbol{y}^{(i)} = \begin{bmatrix} f^{(i)}\left(\boldsymbol{x}_{t_1}\right) \\ \vdots \\ f^{(i)}\left(\boldsymbol{x}_{t_N}\right) \end{bmatrix} + \begin{bmatrix} w_{t_1}^{(i)} \\ \vdots \\ w_{t_N}^{(i)} \end{bmatrix} = \boldsymbol{f}^{(i)}(X) + \boldsymbol{w}^{(i)},$$

where $\boldsymbol{w}^{(i)}$ is i.i.d. Gaussian noise with standard deviation σ_i, while $f^{(i)}(\cdot)$ is an unknown function modeled a priori as a GP, namely, $f^{(i)}(\cdot) \sim N(0, K^{(i)}(X, X))$. The covariance matrix $K^{(i)}(X, X)$ is defined through a kernel function $k^{(i)}(\cdot, \cdot)$. Specifically, the covariance between $f^{(i)}(\boldsymbol{x}_{t_j})$ and $f^{(i)}(\boldsymbol{x}_{t_k})$, i.e., the element of $K^{(i)}(X, X)$ at row j and column l, is equal to $k^{(i)}(\boldsymbol{x}_{t_j}, \boldsymbol{x}_{t_k})$. Exploiting the properties of Gaussian distributions, it can be proven that the posterior distribution of $f^{(i)}$ given $\mathcal{D}$ in a general input location $\boldsymbol{x}_*$ is Gaussian [74]. Then, the maximum a posteriori estimator corresponds to the mean, which is given by the following expression

$$\hat{f}^{(i)}(\boldsymbol{x}_*) = K^{(i)}\left(\boldsymbol{x}_*, X\right)\boldsymbol{\alpha}^{(i)}, \tag{5.4}$$

where

$$\begin{aligned} \boldsymbol{\alpha}^{(i)} &= \left(K^{(i)}\left(X, X\right) + \sigma_i^2 I\right)^{-1}\boldsymbol{y}^{(i)}, \\ K^{(i)}(\boldsymbol{x}_*, X) &= \left[k^{(i)}(\boldsymbol{x}_*, \boldsymbol{x}_{t_1}) \dots k^{(i)}(\boldsymbol{x}_*, \boldsymbol{x}_{t_N})\right]. \end{aligned}$$

Different solutions proposed in the literature can be grouped roughly based on the definition of the GP prior. In this chapter, we will consider two black-box approaches, where the prior is defined without exploiting prior information about the physical model.

Squared Exponential (SE). The SE kernel [74, 83], defines the covariance between samples based on the distance between GP inputs, and it is defined by the following expression

$$k_{SE}(\boldsymbol{x}_{t_j}, \boldsymbol{x}_{t_l}) = \lambda^2 e^{-\frac{1}{2}\left\|\boldsymbol{x}_{t_j} - \boldsymbol{x}_{t_l}\right\|^2_{\Lambda^{-1}}}; \tag{5.5}$$

λ and Λ are the kernel hyper-parameters. The first is a scaling factor, and the second is a positive definite matrix, which defines the norm used to compute the distance between inputs. A common choice consists in considering Σ to be diagonal, with the positive diagonal elements named length-scales.

Geometrically Inspired Polynomial (GIP). The GIP kernel has been recently introduced in [60]. This kernel is based on the property that the dynamics equations in (5.1) are a polynomial function in a proper transformation of the GP input, fully characterized only by the type of each joint. Specifically, $\boldsymbol{q}$ is mapped in $\tilde{\boldsymbol{q}}$, the vector composed by the concatenation of the components associated with a prismatic joint and the sines and cosines of the revolute coordinates. As proved in [60], the inverse dynamics in (5.1) is a polynomial function in $\ddot{\boldsymbol{q}}$, $\dot{\boldsymbol{q}}$ and $\tilde{\boldsymbol{q}}$, where the elements of $\ddot{\boldsymbol{q}}$ have maximum relative degree of one, whereas the ones of $\dot{\boldsymbol{q}}$ and $\tilde{\boldsymbol{q}}$ have maximum relative degree two. To exploit this property, the GIP kernel is defined through the sum and the product of different polynomial kernels [30], hereafter denoted as $k_P^{(p)}(\cdot,\cdot)$, where p is the degree of the polynomial kernel. In particular, we have

$$k_{GIP}(\boldsymbol{x}_{t_j}, \boldsymbol{x}_{t_l}) = \left(k_P^{(1)}(\ddot{\boldsymbol{q}}_{t_j}, \ddot{\boldsymbol{q}}_{t_l}) + k_P^{(2)}(\dot{\boldsymbol{q}}_{t_j}, \dot{\boldsymbol{q}}_{t_l})\right) k_Q(\tilde{\boldsymbol{q}}_{t_j}, \tilde{\boldsymbol{q}}_{t_l}), \tag{5.6}$$

where, in its turn, k_Q is given by the product of polynomial kernels with degree two, see [60] for all the details. In this way, the GIP kernel allows defining a regression problem in a finite-dimensional function space where (5.1) is contained, leading to better data efficiency in comparison with the SE kernel.

5.2 Dynamics Components Estimation from Inverse Dynamics Model

In this section, we describe how it is possible to obtain estimates of the different contributions in the left-hand side of (5.1) when adopting GPR to identify the inverse dynamics; in particular, we discuss the computation of gravitational contributions, inertial contri-

butions, and $\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})$. We assume that a distinct GP is used for each of the n degree of freedom, and we denote by $\hat{f}^{(i)}(\cdot)$, $i = 1 \dots n$, the estimator of the i-th joint torque obtained applying (5.4). For convenience, from here on, we will point out explicitly the different components of the GP input, namely, the input of the $\hat{f}^{(i)}$ will be $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$ instead of $\boldsymbol{x}$, which comprises the concatenation of $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$. It is worth mentioning that the proposed approach is inspired by the strategy adopted in Newton-Euler algorithms, see [31].

5.2.1 Gravitational contribution

As shown in (5.1), the torques due to the gravitational contributions account for all the terms that depend only on $\boldsymbol{q}$. Consequently, to obtain $g^{(i)}(\boldsymbol{q})$, i.e., the estimate of the i-th gravitational contribution in the configuration $\boldsymbol{q}$, we evaluate $\hat{f}^{(i)}$ by setting $\dot{\boldsymbol{q}} = \boldsymbol{0}$, $\ddot{\boldsymbol{q}} = \boldsymbol{0}$. Then, the estimate of $\boldsymbol{g}(\boldsymbol{q})$ is

$$\hat{\boldsymbol{g}}(\boldsymbol{q}) = \begin{bmatrix} \hat{g}^{(1)}(\boldsymbol{q}) \\ \vdots \\ \hat{g}^{(n)}(\boldsymbol{q}) \end{bmatrix} = \begin{bmatrix} \hat{f}^{(1)}(\boldsymbol{q}, \boldsymbol{0}, \boldsymbol{0}) \\ \vdots \\ \hat{f}^{(n)}(\boldsymbol{q}, \boldsymbol{0}, \boldsymbol{0}) \end{bmatrix}. \tag{5.7}$$

5.2.2 Inertial contributions

The inertial contributions, i.e., $B(\boldsymbol{q})\ddot{\boldsymbol{q}}$, accounts for all the contributions that depend simultaneously on $\boldsymbol{q}$ and $\ddot{\boldsymbol{q}}$. Consequently, to estimates these contributions, we evaluate the GP models in $(\ddot{\boldsymbol{q}}, \boldsymbol{0}, \boldsymbol{q})$, and subtract the gravitational contribution defined and computed previously. In particular, to obtain $\hat{B}_{ij}(\boldsymbol{q})$, i.e., the estimate of the $B(\boldsymbol{q})$ element in position (i, j), we set all the accelerations to zero, except for the j-th component. Denoting with $\boldsymbol{1}_j$ the vector with all elements equal to zero except for the j-th element, which, instead, is equal to one, we have

$$\hat{B}_{ij}(\boldsymbol{q}) = \hat{f}^{(i)}(\boldsymbol{q}, \boldsymbol{0}, \boldsymbol{1}_j) - \hat{g}^{(i)}(\boldsymbol{q}). \tag{5.8}$$

5.2.3 Estimation of $\boldsymbol{n}(\boldsymbol{q})$

The vector $\boldsymbol{n}(\boldsymbol{q})$ collects all the contributions that do not depend on $\ddot{\boldsymbol{q}}$. Then, $n^{(i)}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, i.e., the estimate of the i-th component of $\boldsymbol{n}(\boldsymbol{q})$, is computed by evaluating the i-th GP

model setting $\ddot{\boldsymbol{q}} = \mathbf{0}$. Then, we have

$$\hat{\boldsymbol{n}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{bmatrix} \hat{n}^{(1)}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \\ \vdots \\ \hat{n}^{(n)}((\boldsymbol{q}, \dot{\boldsymbol{q}})) \end{bmatrix} = \begin{bmatrix} \hat{f}^{(1)}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \mathbf{0}) \\ \vdots \\ \hat{f}^{(n)}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \mathbf{0}) \end{bmatrix}. \tag{5.9}$$

5.3 Feedback Linearization Control Based on Gaussian Process Model

In this section, we describe the two GP-based feedback linearization controllers implemented. The first implementation aims at estimating directly an approximation of (5.2) using the GP models, whereas the second computes the approximation of (5.2) by estimating $B(\boldsymbol{q})$ and $\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ using the expressions derived in Section 5.2.

5.3.1 GP-FL

In this approach, hereafter denoted as *GP Feedback Linearization control* (GP-FL), the control input is selected to be directly the estimate of (5.2b). The estimate of (5.2b) at time t is obtained by evaluating the n GP models with GP-input given by the concatenation of $\boldsymbol{q}_t$, $\dot{\boldsymbol{q}}_t$ and $\boldsymbol{a}_t = \ddot{\boldsymbol{r}}_t + K_p \boldsymbol{e}_t + K_d \dot{\boldsymbol{e}}_t$. Then, referring to the notation previously introduced, we have

$$\boldsymbol{\tau}_t = \hat{\boldsymbol{f}}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{a}_t) = \begin{bmatrix} \hat{f}^{(1)}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{a}_t) & \dots & \hat{f}^{(n)}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{a}_t) \end{bmatrix}^T. \tag{5.10}$$

The overall GP-FL control architecture is schematized in Figure 5.2.

5.3.2 GP-FL-DCE

The second approach, named *GP Feedback Linearization control with Dynamics Components Estimation* (GP-FL-DCE), computes the control input based on (5.2) and the estimation of $B(\boldsymbol{q})$ and $\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ obtained with the GP input. First, the elements of the inertia matrix and the estimates of $\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ are computed by applying, respectively, (5.8) and (5.9). Then, the input is

$$\boldsymbol{\tau}_t = \hat{B}(\boldsymbol{q}_t)\boldsymbol{a}_t + \hat{\boldsymbol{n}}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t) = \begin{bmatrix} \hat{B}_{11}(\boldsymbol{q}) & \dots & \hat{B}_{1n}(\boldsymbol{q}) \\ \vdots & \vdots & \vdots \\ \hat{B}_{n1}(\boldsymbol{q}) & \dots & \hat{B}_{nn}(\boldsymbol{q}) \end{bmatrix} \boldsymbol{a}_t + \begin{bmatrix} \hat{n}^{(1)}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \\ \vdots \\ \hat{n}^{(n)}((\boldsymbol{q}, \dot{\boldsymbol{q}})) \end{bmatrix}, \tag{5.11}$$

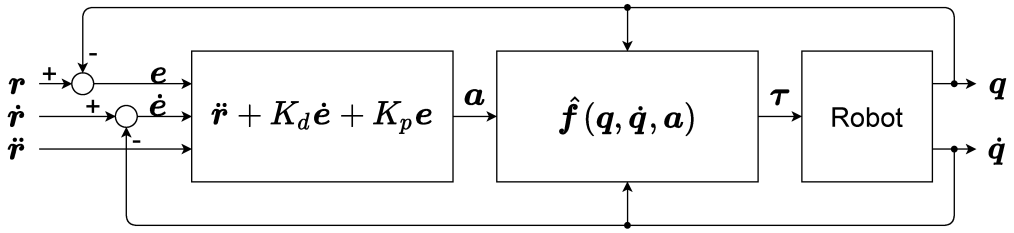


Figure 5.2: GP-FL control diagram. $\boldsymbol{q}$, $\boldsymbol{r}$, and $\boldsymbol{e}$ are the joints position, the reference trajectory, and the tracking error.

where, as before, $\boldsymbol{a}_t = \ddot{\boldsymbol{r}}_t + K_p \boldsymbol{e}_t + K_d \dot{\boldsymbol{e}}_t$. The overall GP-FL-DCE control architecture is the reported in Figure 5.1b, where estimates $\hat{B}(\boldsymbol{q})$ and $\hat{\boldsymbol{n}}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ are obtained exploiting the trained GP model.

5.4 Experiments in Simulation

Experiments have been carried out in PyBullet [23], simulating a KUKA LBR iiwa, which is a 7-DoF collaborative manipulator[1]. The system was controlled at 1000 HZ. Position, velocity, and torques of the joints are directly provided by the simulator. The accelerations needed for model identification were computed offline by means of acausal numerical differentiation of the velocities. Specifically, we applied the central difference approximation, namely, the acceleration of the joints at time t is approximated with $\ddot{\boldsymbol{q}}_t = (\dot{\boldsymbol{q}}_{t+1} - \dot{\boldsymbol{q}}_{t-1})/(2T_s)$, where T_s is the sampling time.

The remainder of this section is organized as follows. First, we compare the accuracy of GP models obtained with the SE and GIP kernels. Second, the control performance of GP-FL and GP-FL-DCE is compared on a trajectory tracking problem, with initial tracking error equal to zero and varying the kernel choice. Finally, the two strategies are tested on the same trajectory tracking problem, in the presence of initial tracking errors for all joints.

5.4.1 Model learning performance

To train and test the GP models obtained with the SE and GIP kernel, we collected two data sets, denoted by $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$. The first data set, $\mathcal{D}_{train}$, was used to derive the GP estimators (5.4), after optimizing the kernel hyper-parameters by marginal likelihood maximization [74]. The second data set, $\mathcal{D}_{test}$, was used to compare the performance of the two GP estimators. Both data sets were collected by employing a hand-tuned PD controller to track a random reference trajectory. For each joint, the reference trajectory

[1] A video of the experiments is available at `https://youtu.be/ehy8iDRGIDo`

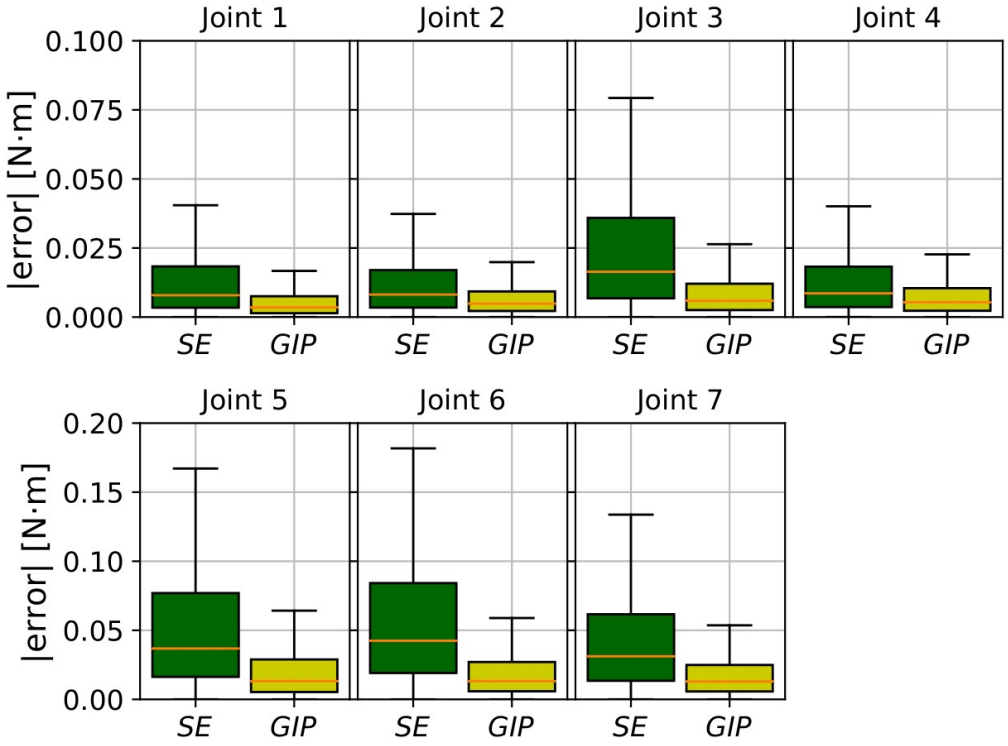


Figure 5.3: Boxplots of the absolute errors obtained in $\mathcal{D}_{test}$ with the SE and GIP kernels. In the table below, we report the nMSE percentages.

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
SE	3.99	0.48	4.22	0.88	7.95	10.86	4.91
GIP	0.42	0.12	0.55	0.18	1.21	1.80	1.05

was Gaussian noise filtered with a second-order low-pass filter (with cutoff frequency 1 Hz). The length of the trajectory was 50 seconds, resulting in 50,000 samples. To limit the computational complexity of (5.4), the collected samples were down-sampled with a constant rate of 10, obtaining 5,000 samples for each dataset.

In Figure 5.3, we visualize the distribution of the absolute value of the errors obtained in $\mathcal{D}_{test}$ with the two GP estimators. Moreover, in the table below Figure 5.3, we reported the normalized Mean Squared Error (nMSE), namely, the ratio between the mean squared error and the variance of the correspondent joint torques, expressed as a percentage. As already showed in [60], for all joints, the estimator based on the GIP kernel outperforms the one based on the SE kernel, showing better data efficiency and generalization.

5.4.2 Trajectory tracking without initial tracking error

In the first control experiment, the GP-FL and GP-FL-DCE controllers based on the two models were tested on the same trajectory tracking problem. For each dof, $j = 1, \ldots, 7$,

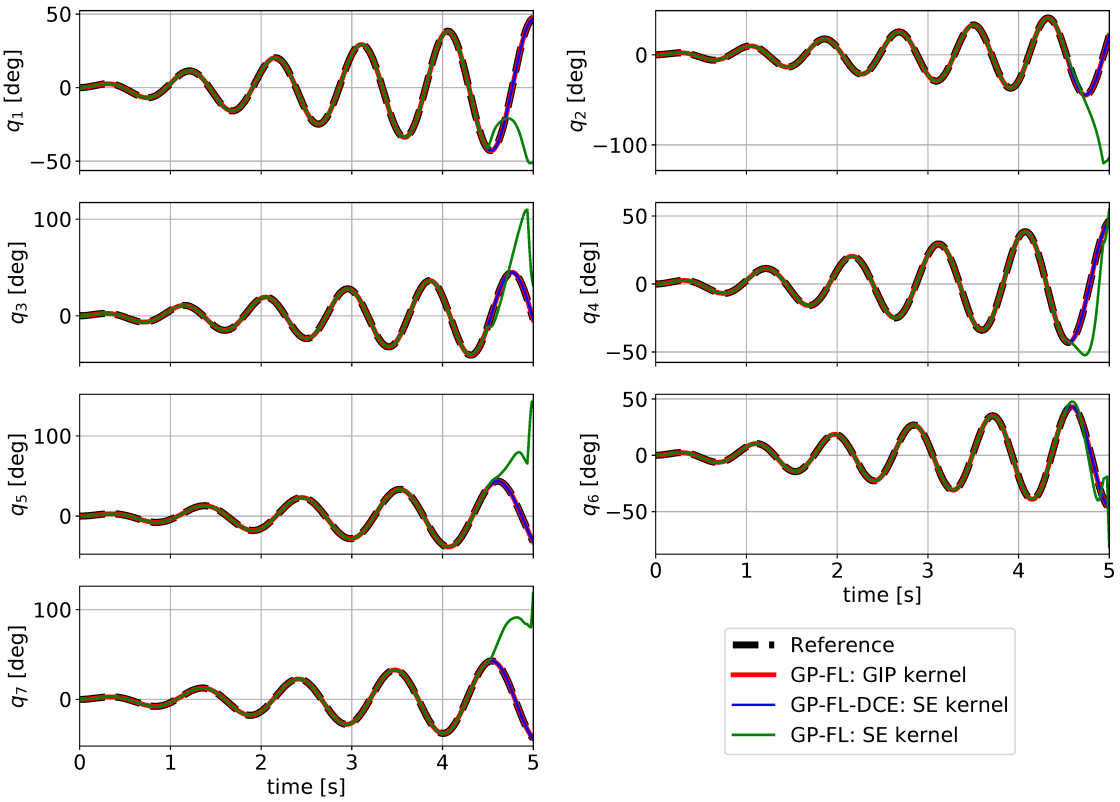


Figure 5.4: Joints trajectories obtained with GP-FL (with the SE and GIP kernels), and GP-FL-DCE (with the SE kernel) in the experiment of Section 5.4.2.

the reference joint position was given by $r_t^{(j)} = 0.165\, t\, sin(2\pi F_j t)$, where the frequencies F_j were randomly sampled from $\mathcal{N}(0.5, 1)$. The controller gains are selected following the considerations reported in Section 5.1, $K_p = \omega^2 I$ and $K_d = 2\zeta\omega I$, with $\omega = 100$ and $\zeta = 2$. The control horizon was set to 5 seconds, and the initial tracking error was zero. In Figure 5.4 and 5.5, the evolution of the joint angles and control torques obtained by GP-FL with SE and GIP kernel, and GP-FL-DCE with SE kernel are reported, respectively.

First, we discuss the performance obtained using the model based on the SE kernel. It can be noticed that the GP-FL controller with the SE kernel works properly when the amplitudes of the reference oscillations are low, but it starts to fail suddenly towards the end of the control horizon, when zero torques are commanded to all joints. This observation suggests that the GP-FL scheme evaluates the GP model in unexplored regions, where predictions are equal to the prior mean [74], which is zero. This is due to the large magnitude of $\boldsymbol{a}_t$, which grows with the tracking error, and becomes significantly different from the accelerations seen during training. Instead, GP-FL-DCE with the SE

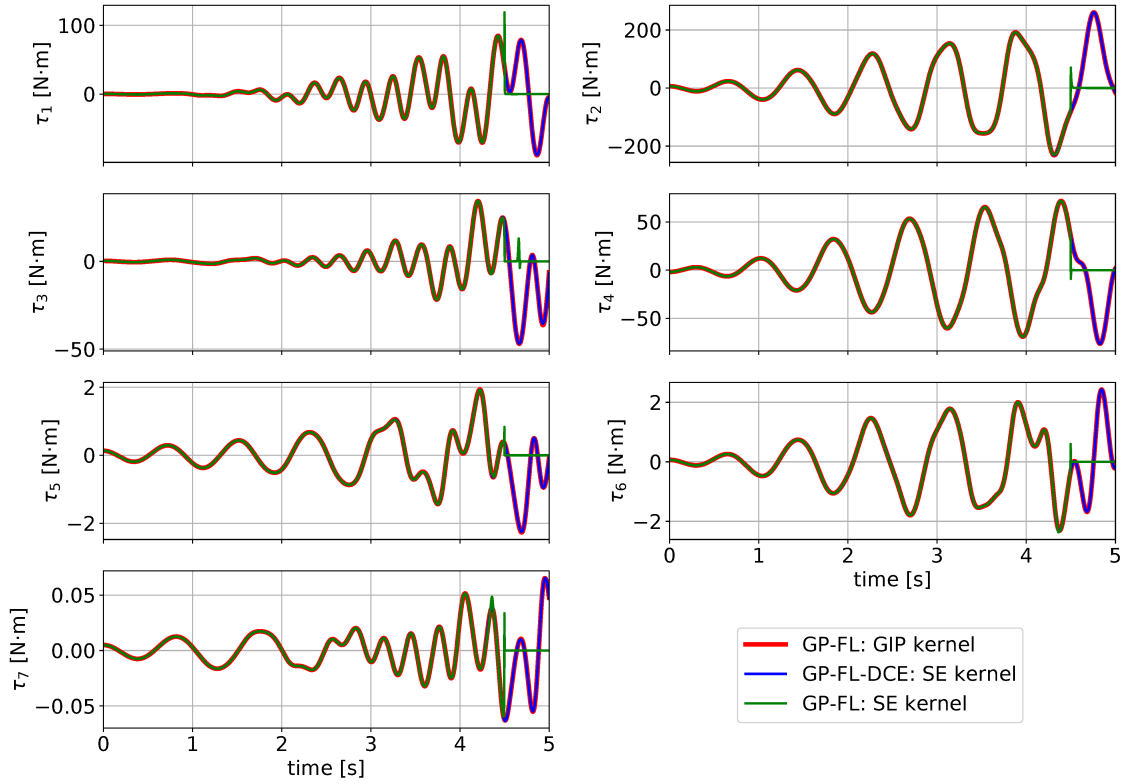


Figure 5.5: Joint torques obtained with GP-FL (with the SE and GIP kernels), and GP-FL-DCE (with the SE kernel) in the experiment of Section 5.4.2.

kernel is able to track the reference trajectory, demonstrating better robustness compared to the GP-FL. This robustness is likely achieved thanks to the estimation of the individual components of the dynamics. Indeed, even though the robot is far from the configurations seen during training, the GP model based on SE provides sufficiently accurate estimates of $B(\boldsymbol{q})$ and $\boldsymbol{n}(\boldsymbol{q})$, which results in keeping the robot close to the reference.

Thanks to the better generalization of the GIP kernel, the GP-FL controller based on the GIP kernel is more robust and is always able to track the desired reference trajectory, also in unexplored areas of the state space. The performance of GP-FL-DCE with the GIP kernel has not been reported, since the trajectory obtained by GP-FL-DCE with the GIP kernel is the same of GP-FL with the GIP kernel. This is due to the definition of the GIP kernel, which is closer to the physics of the system, and already encodes the linear dependencies of the torques on the acceleration of the joints.

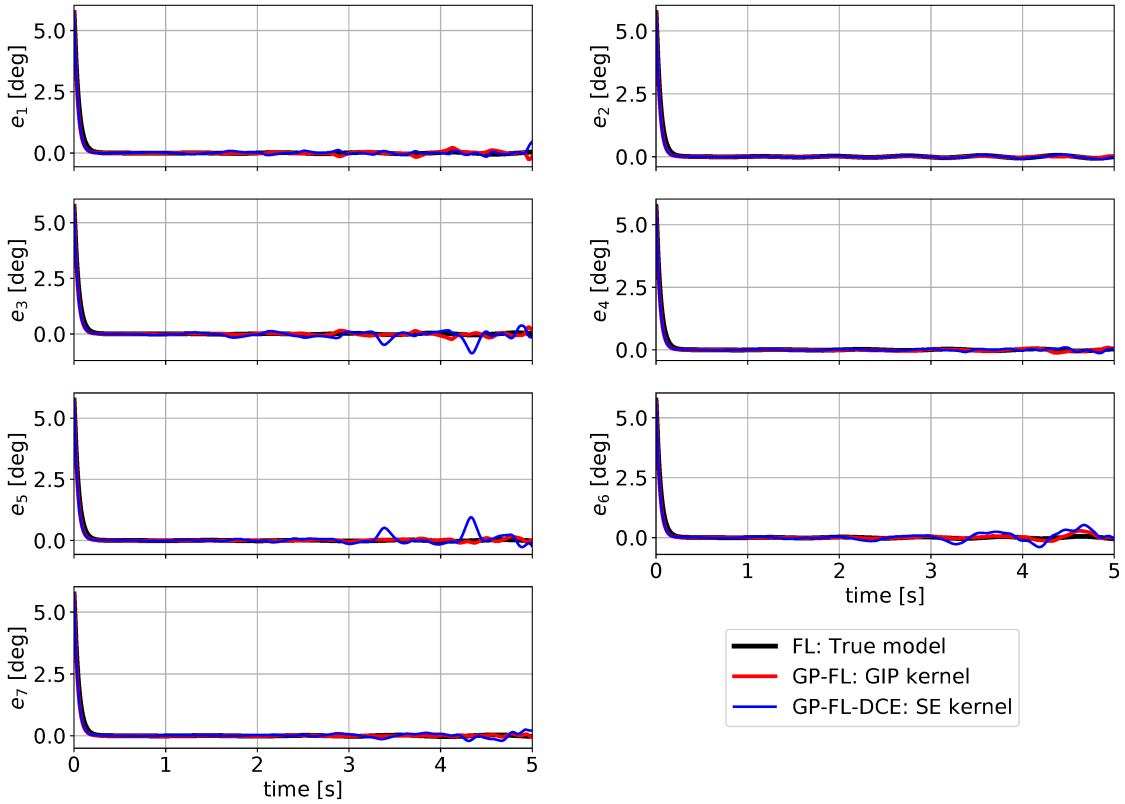


Figure 5.6: Tracking errors obtained with feedback linearization based on the true model, GP-FL (with the GIP kernel), and GP-FL-DCE (with the SE kernel) in the experiment of Section 5.4.3.

5.4.3 Trajectory tracking with initial tracking error

In this experiment, we tested the controllers on the same reference trajectory as in the previous experiment, considering also the presence of initial tracking errors. For all joints, we considered an initial error of 5.73 [deg] (1 [rad]). The obtained behavior confirmed the observations from the previous experiment.

The GP-FL scheme with the SE kernel is not effective. In fact, the initial error makes the magnitude of the $\boldsymbol{a}_t$ term large, leading to considerable distances from accelerations observed during training. As already seen in the previous experiment, this fact makes GP-FL provide null torques, causing the control failure from the beginning. In Figure 5.6, we plotted the tracking errors obtained by GP-FL-DCE with the SE kernel and GP-FL with the GIP kernel, as well as the one obtained by a feedback linearization control based on the true model (we omitted the results of GP-FL with the SE kernel as it cannot control the robot).

For all three estimators, the main dynamics of the tracking error follows the exponential

behavior described in (5.3). Significant differences between the tracking error evolution can be appreciated only at steady state, where the controllers based on GP models are subject to limited oscillations around zero, with absolute value lower than 1 [deg], and growing with the amplitude of the reference trajectories. These errors are due to model inaccuracies, which becomes more relevant when the reference trajectories cross regions that are far from the distribution of the training samples. The errors are higher for the controller based on the SE kernel. This is in accordance with the considerations presented in Section 5.4.1, where we highlight that the model based on GIP is more accurate. In particular, the tracking errors at steady state are higher in joint 3, 4, 6, and 7, which are the ones where the GP estimator is less accurate, as confirmed by the nMSE obtained in the experiment of Section 5.4.1.

5.5 Conclusion

In this chapter, we analyzed the implementation of feedback linearization control schemes based on GP models of the inverse dynamics. We considered two strategies. The first computes the control input directly with the GP model, whereas the second computes the input after estimating the individual components of the dynamics, in particular, the inertia matrix and the torques independent from accelerations. The two strategies were compared on a trajectory tracking problem with a simulated 7-DoF manipulator, varying also the kernel choice; we considered the SE and GIP kernels. Results show that the second implementation is more robust w.r.t. the kernel choice and model inaccuracies. Moreover, as regards the choice of kernel, the obtained performance shows that the use of a structure kernel, such as the GIP kernel, is advantageous, resulting in good performance for both feedback linearization control implementations.

Part III

Gaussian Processes for Modeling Cloth Dynamics

6

Modeling Robotic Cloth Manipulation using Gaussian Processes

Robotic cloth manipulation has a wide range of applications, from textile industry to assistive robotics [10, 64, 54, 82, 41, 13]. However, the complexity of cloth behaviour results in a high uncertainty in the state transition given a certain action. This uncertainty is what makes cloth manipulation much harder than manipulating rigid objects. Intuitively, using a cloth model is the solution to reduce such uncertainty by knowing an approximation of cloth dynamics. In literature, we can find several cloth models that simulate the internal cloth state [92, 6, 66]. However, fitting those models to real data can be a complex task. Moreover, such models need not only to behave similarly enough to the cloth garment, but to have a tractable dimension, for computational reasons. Imagine a piece of cloth e.g. a square towel. If we build an 8×8 cloth mesh, and characterize the three Cartesian components of each point, the cloth state results in a 192-dimensional manifold. Such dimensionality is unmanageable, not only in terms of computational costs, but also for building a tractable state-action space policy. Such is the case of [7], where simulated results are obtained after hours of computations. Therefore, Dimensionality Reduction (DR) methods become necessary for this approach.

In [21], linear DR techniques were used for learning cloth manipulation by biasing the latent space projection with each execution's performance. Nonlinear methods, such

as Gaussian Process Latent Variable Models (GPLVM) [55] have also been applied for this purpose. In [51], GPLVM was employed to project task-specific motor-skills of the robot onto a much smaller state representation, whereas in [37] GPLVM was also used to represent a robot manipulation policy in a latent space, taking contextual features into account. However, these approaches focus the dimensionality reduction in the robot action characterization, rather than in the manipulated object's dynamics. Instead, in [52] the same DR technique was applied to learn a latent representation of the cloth state from point cloud observations taken with a motion capture system and depth sensors. However, such approach did not consider the dynamics of the cloth handling task, and its applicability is limited to a quasi-static manipulation.

In this work, we assume to have recorded data from several cloth motions, as a time-varying mesh of points. This data will often come from the processed RGB-D data of a camera. To fit such data into a tractable dynamical model, we consider Gaussian Process Dynamical Models (GPDM), first introduced in [96], which are an extension of the GPLVM structure explicitly oriented to the analysis of high-dimensional time series. GPDMs have been applied in several different fields, from human motion tracking [95, 94] to dynamic texture modeling [99]. In the context of cloth manipulation, GPDMs were adopted in [50] to learn a latent model of the dynamics of a cloth handling task. However, this framework, as it stands, lacks in its structure a fundamental component to correctly describe the dynamics of a system, namely control actions, limiting generalization capacity.

Therefore, we propose an extension of the GPDM structure, that takes into account the influence of external control actions on the modeled dynamics. We call it *Controlled Gaussian Process Dynamical Models* (CGPDMs). In this new version of the model, control actions directly affect the dynamics in the latent space. Thus, a CGPDM, trained on a sufficiently diverse set of interactions, is able to predict the effects of control actions never experienced before inside a space of reduced dimensions, and then reconstruct high-dimensional motions by projecting the latent state trajectories into the observation space. We tested the applicability of such solution in both a simulated and a real cloth manipulation scenario. CGPDM has proved capable of fitting different types of cloth movement, and predict the results of sequences of control actions never seen during training.

Previous GPDM applications [96, 95, 94] were based on Gaussian Processes (GPs) equipped with isotrophic squared exponential (SE) and homogeneous linear kernel functions. The limited number of hyper-parameters of such GPs could limit the generalization properties of the overall model. In this work, we study the effects that a richer pa-

rameterization in both latent and dynamical maps could possibly have on prediction accuracy and generalization capabilities. In particular, we compare two distinct CGPDM implementations: (i) the *lowly-parameterized* CGPDM is a direct extension of the standard GPDM's dynamical and latent maps to the case of a controlled dynamical system; (ii) the *highly-parameterized* CGPDM, instead, is characterized by the presence of a higher number of hyper-parameters. In the last case, we employed the SE kernels with automatic relevance determination (ARD) [65] and inhomogeneous linear kernels, together with tunable scaling factors in the dynamical map. Despite these kernels are a common choice in GP Regression (GPR) practice, they have not yet been applied in the context of GPDM. In fact, to the best of our knowledge, only [99] proposed to use kernels different from the naive ones adopted in the original implementation [96], but it introduced a multi-kernel structure only for the GPs modeling the dynamics, without modifying the latent map.

The remainder of the chapter is structured as follows. Preliminaries are given in Sec. 6.1. CGPDM framework is presented in Sec. 6.2. Sec. 6.3 treats its training and explains how to make predictions, in both latent and observation spaces. Experimental results on a simulated robotic cloth manipulation scenario are reported in Sec. 6.4, while the application of CGPDM on a real setup is described in Sec. 6.5. Sec. 6.6 draws the conclusions.

6.1 Preliminaries: GPLVM and GPDM

GP regression has been widely applied as a data-driven tool for dynamical system identification [49], usually describing each state by its own GP. Nevertheless, such approach is unfeasible when dealing with high-dimensional systems, due to the high computational demands. Thus, DR strategies must be included. GPLVM [55] and its variations [59] emerged as feature extraction methods that can be used as multiple-output GPR models. In this way, these models, under a DR perspective, associate and learn low-dimensional representations of higher-dimensional observed data, assuming that observed variables are determined by the latent ones. Intuitively, a GPLVM assumes the high-dimensional observations to be the target values of an ensamble of GPs. The associated input locations are interpreted as the latent states, considering them as hyper-parameters of the overall model. Hence, the GPLVM provide, as a result of an optimization, a mapping from the latent space to the observation space, together with a set of latent variables representing the observed values. However, GPLVMs are not explicitly thought to deal with data from time series, where observations at different time

steps are connected by some form of dynamics.

Thus, [96] first introduced Gaussian Process Dynamical Models (GPDM) , an extension of the GPLVM structure explicitly oriented to the analysis of high-dimensional time series. A GPDM entails essentially two stages: (i) a latent mapping that projects high-dimensional observations to a low-dimensional latent space (6.1); (ii) a discrete-time Markovian dynamics that captures the evolution of the time series inside the reduced latent space (6.2). GPs are used to model both the latent map, as in GPLVMs, and the undergoing dynamics transition function. GPDMs are then defined by

$$\boldsymbol{y}_t = g(\boldsymbol{x}_t) + \boldsymbol{n}_{y,t}, \tag{6.1}$$

$$\boldsymbol{x}_{t+1} = h(\boldsymbol{x}_t) + \boldsymbol{n}_{x,t}, \tag{6.2}$$

where $\boldsymbol{y}_t$ is the high-dimensional observation vector and $\boldsymbol{x}_t$ represents the latent state, at time step t. Here, $\boldsymbol{n}_{y,t}$ and $\boldsymbol{n}_{x,t}$ are two zero-mean isotropic white Gaussian noise processes, while g and h indicate the unknown functions characterizing the two maps.

6.2 Controlled GPDM

Let us consider a system governed by an unknown dynamics. At each time step t, it is possible to influence it by applying control actions $\boldsymbol{u}_t \in \mathbb{R}^E$ and getting an observation $\boldsymbol{y}_t \in \mathbb{R}^D$. For high-dimensional observation spaces, it could be unfeasible to directly model the evolution of a sequence of observations in response to a series of inputs. For instance, in the case of a robot moving a piece of cloth, we can consider as control actions $\boldsymbol{u}_t$ the instantaneous movement of the end-effector, while the observations $\boldsymbol{y}_t$ could be the coordinates of a mesh of material points, representing the cloth configuration. In this context, it could be convenient to capture the dynamics of the system in a low-dimensional latent space $\mathbb{R}^d$, with $d << D$. Let $\boldsymbol{x}_t \in \mathbb{R}^d$ be the latent state associated with $\boldsymbol{y}_t$. We propose to use a variation of the GPDM that keeps into account the influence of control actions, while maintaining the dimensionality reduction properties of the original model. We call it *Controlled Gaussian Process Dynamical Model* (CGPDM).

A CGPDM consists of a latent map (6.3) projecting observations $\boldsymbol{y}_t$ into latent states $\boldsymbol{x}_t$, and a dynamics map (6.4) that describes the evolution of the latent state $\boldsymbol{x}_t$, subject to control action $\boldsymbol{u}_t$.

$$\boldsymbol{y}_t = g(\boldsymbol{x}_t) + \boldsymbol{n}_{y,t}, \tag{6.3}$$

$$\boldsymbol{x}_{t+1} - \boldsymbol{x}_t = h(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{n}_{x,t}. \tag{6.4}$$

Differently from original GPDM (6.2), control actions have influence on the CGPDM

transition function (6.4). On the other hand, latent map (6.3) is identical to (6.1) because control actions should not affect the dimensionality reduction process. Note that we consider $\boldsymbol{x}_{t+1} - \boldsymbol{x}_t$ to be the output of the CGPDM dynamic map, [95] suggested that this choice can help to improve the smoothness of latent trajectories. In the following, we report how to model (6.3) and (6.4) by means of GPs.

6.2.1 Latent variable mapping

Each component of the observation vector $\boldsymbol{y}_t = [y_t^{(1)}, \dots, y_t^{(D)}]^T$ can be modeled a priori as a zero-mean GP that takes as input $\boldsymbol{x}_t$, for $t = 1, \dots, N$.

Let $Y = [\boldsymbol{y}_1, \dots, \boldsymbol{y}_N]^T \in \mathbb{R}^{N \times D}$ be the matrix that collects the set of N observations, and $X = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_N]^T \in \mathbb{R}^{N \times d}$ be the matrix of associated latent states. We denote with $Y_{:,j}$ the vector containing the j-th components of all the N observations, for $j = 1, \dots, D$.

Then, we have $Y_{:,j} \sim \mathcal{N}(0, K_y^{(j)}(X))$ for $j = 1, \dots, D$. The covariance matrix $K_y^{(j)}(X)$ is defined through a kernel function $k_y^{(j)}(\cdot, \cdot)$. Specifically, for any two observations $r, s = 1, \dots, N$, the covariance between their j-th components $y_r^{(j)}$ and $y_s^{(j)}$, i.e. the element of $K_y^{(j)}(X)$ at row r and column s, is equal to $k_y^{(j)}(\boldsymbol{x}_r, \boldsymbol{x}_s)$. The probability distribution of $Y_{:,j}$ is given by

$$p(Y_{:,j}|X) = \frac{\exp\left(-\frac{1}{2} Y_{:,j}^T \left(K_y^{(j)}(X)\right)^{-1} Y_{:,j}\right)}{\sqrt{(2\pi)^N |K_y^{(j)}(X)|}}.$$

If we assume that the D observation components are independent variables, the probability over the whole set of observations can be expressed by the product of the D GPs. In addition, if we choose identical kernel functions for each dimension, with the only difference given by a scaling factor, i.e. $k_y^{(j)}(\cdot, \cdot) = w_{y,j}^{-2} k_y(\cdot, \cdot)$ for $j = 1, \dots, D$, the joint likelihood over the whole set of observations is given by

$$p(Y|X) = \frac{|W_y|^N \exp\left(-\frac{1}{2}\mathrm{tr}\left((K_y(X))^{-1} Y W_y^2 Y^T\right)\right)}{\sqrt{(2\pi)^{ND} |K_y(X)|^D}}, \tag{6.5}$$

where $W_y = \mathrm{diag}(w_{y,1}, \dots, w_{y,D})$, $K_y(X)$ is the covariance matrix built employing only $k_y(\cdot, \cdot)$, i.e. the common part of kernel functions. Independence assumption may be relaxed by applying, for instance, coregionalization models [100], at the cost of greater computational demands, that might be unfeasible in our high-dimensional context.

In previous works on GPDMs [96, 95, 94], the GPs of the latent map are usually

equipped with a simple isotrophic form of the SE kernel,

$$k'_y(\boldsymbol{x}_r, \boldsymbol{x}_s) = \exp\left(-\frac{\beta_1}{2}||\boldsymbol{x}_r - \boldsymbol{x}_s||^2\right) + \beta_2^{-1}\delta(\boldsymbol{x}_r, \boldsymbol{x}_s), \tag{6.6}$$

parameterized only by β_1 and β_2, where $\delta(\boldsymbol{x}_r, \boldsymbol{x}_s)$ indicates the Kronecker delta.

Instead, we adopt the richer ARD structure for this kernel, characterized by the presence of length-scales that can weight differently each component of the latent state:

$$k_y(\boldsymbol{x}_r, \boldsymbol{x}_s) = \exp\left(-\frac{1}{2}||\boldsymbol{x}_r - \boldsymbol{x}_s||_{\Lambda_y^{-1}}\right) + \sigma_y^2\delta(\boldsymbol{x}_r, \boldsymbol{x}_s). \tag{6.7}$$

$\Lambda_y^{-1} = \text{diag}(\lambda_{y,1}^{-2}, \dots, \lambda_{y,D}^{-2})$ is a positive definite diagonal matrix, which weights the norm used in the SE function, and σ_y^2 is the variance of the isotropic noise in (6.3). The trainable hyper-parameters of the latent map are then $\theta_y = \{w_{y,1}, \dots, w_{y,D}, \lambda_{y,1}, \dots, \lambda_{y,D}, \sigma_y\}$.

6.2.2 Dynamics mapping

Similarly to Section 6.2.1, we can model a priori each component of the latent state difference $\boldsymbol{x}_{t+1} - \boldsymbol{x}_t = [x_{t+1}^{(1)} - x_t^{(1)}, \dots, x_{t+1}^{(d)} - x_t^{(d)}]^T$ as a zero-mean GP that takes as input the pair $(\boldsymbol{x}_t, \boldsymbol{u}_t)$, for $t = 1, \dots, N-1$.

Let $X = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_N]^T \in \mathbb{R}^{N\times d}$ be the matrix collecting the set of N latent states, we can denote by $X_{r:s,i}$ the vector of the i-th components from time step r to time step s, with $r, s = 1, \dots, N$. We indicate the vector of differences between consecutive latent states along their i-th component with $\Delta_{:,i} = (X_{2:N,i} - X_{1:N-1,i}) \in \mathbb{R}^{N-1}$. $\Delta = [\Delta_{:,1}, \dots, \Delta_{:,d}] \in \mathbb{R}^{(N-1)\times d}$ is the matrix that collects differences along all the components. Finally, we compactly represent the GP input of the dynamic model as $\tilde{\boldsymbol{x}}_t = [\boldsymbol{x}_t^T, \boldsymbol{u}_t^T]^T \in \mathbb{R}^{d+E}$, and refer to the the matrix collecting $\tilde{\boldsymbol{x}}_t$ for $t = 1, \dots, N-1$ with $\tilde{X} = [\tilde{\boldsymbol{x}}_1, \dots, \tilde{\boldsymbol{x}}_{N-1}]^T \in \mathbb{R}^{(N-1)\times(d+E)}$.

Thus, we have $\Delta_{:,i} \sim \mathcal{N}(0, K_x^{(i)}(\tilde{X}))$ for $i = 1, \dots, d$. The covariance matrix $K_x^{(i)}(\tilde{X})$ is defined through a kernel function $k_x^{(i)}(\cdot,\cdot)$. Specifically, for any $r, s = 1, \dots, N-1$, the covariance between $x_{r+1}^{(1)} - x_r^{(1)}$ and $x_{s+1}^{(1)} - x_s^{(1)}$, i.e. the element of $K_x^{(i)}(\tilde{X})$ at row r and column s, is equal to $k_x^{(i)}(\tilde{\boldsymbol{x}}_r, \tilde{\boldsymbol{x}}_s)$. The probability density over $\Delta_{:,i}$ is then given by

$$p(\Delta_{:,i}|\tilde{X}) = \frac{\exp\left(-\frac{1}{2}\Delta_{:,i}^T\left(K_x^{(i)}(\tilde{X})\right)^{-1}\Delta_{:,i}\right)}{\sqrt{(2\pi)^N|K_x^{(i)}(\tilde{X})|}}.$$

Assuming that the d latent state components evolve independently, the probability over the whole Δ can be expressed by the product of the d GPs. Again, if we choose identical kernel functions for each dimension, with different scaling factors, i.e. $k_x^{(i)}(\cdot,\cdot) =$

$w_{x,i}^{-2}k_x(\cdot,\cdot)$ for $i=1,\dots,d$, we obtain the following expression for the joint likelihood:

$$p(\Delta|\tilde{X}) = \frac{|W_x|^{N-1}\exp\left(-\frac{1}{2}\text{tr}\left(\left(K_x(\tilde{X})\right)^{-1}\Delta W_x^2\Delta^T\right)\right)}{\sqrt{(2\pi)^{(N-1)d}|K_x(\tilde{X})|^d}}, \tag{6.8}$$

where $W_x = \text{diag}(w_{x,1},\dots,w_{x,d})$, $K_x(\tilde{X})$ is the covariance matrix built employing only $k_x(\cdot,\cdot)$, i.e. the common part of kernel functions.

In standard GPDM [96], dynamic mapping GPs were initially proposed with constant scaling factors $w_{x,i}=1$ for $i=1,\dots,d$, and equipped with the simple kernel resulting from the sum of an isotrophic SE and an homogeneous linear function, characterized only by four trainable parameters:

$$k'_x(\tilde{\boldsymbol{x}}_r,\tilde{\boldsymbol{x}}_s) = \alpha_1\exp\left(-\frac{\alpha_2}{2}||\tilde{\boldsymbol{x}}_r-\tilde{\boldsymbol{x}}_s||^2\right)+\alpha_3\tilde{\boldsymbol{x}}_r^T\tilde{\boldsymbol{x}}_s+\alpha_4^{-1}\delta(\tilde{\boldsymbol{x}}_r,\tilde{\boldsymbol{x}}_s). \tag{6.9}$$

Analogously to what was previously proposed for the latent mapping, we adopted a more complex kernel function, detailed in the following,

$$k_x(\tilde{\boldsymbol{x}}_r,\tilde{\boldsymbol{x}}_s) = \exp\left(-\frac{1}{2}||\tilde{\boldsymbol{x}}_r-\tilde{\boldsymbol{x}}_s||_{\Lambda_x^{-1}}\right)+[\tilde{\boldsymbol{x}}_r^T,1]\Phi[\tilde{\boldsymbol{x}}_s^T1]^T+\sigma_x^2\delta(\tilde{\boldsymbol{x}}_r,\tilde{\boldsymbol{x}}_s). \tag{6.10}$$

$\Lambda_x^{-1} = \text{diag}(\lambda_{x,1}^{-2},\dots,\lambda_{x,d+E}^{-2})$ is a positive definite diagonal matrix, which weights the norm used in the SE component of the kernel. Also $\Phi = \text{diag}(\phi_1^2,\dots,\phi_{d+E+1}^2)$ is a positive definite diagonal matrix that characterizes the linear kernel component. σ_x^2 is the variance of the isotropic noise in (6.4). In comparison to (6.9), the adopted kernel weights differently the various components of the input in both SE and linear part, where the GP input is also extended as $[\tilde{\boldsymbol{x}}_s^T,1]^T$. The trainable hyper-parameters of the dynamic map model are then $\theta_x = \{w_{x,1},\dots,w_{x,d},\lambda_{x,1},\dots,\lambda_{x,d},\phi_1,\dots,\phi_{d+E+1},\sigma_x\}$.

In the following we will refer to the proposed CGPDM structure with kernels (6.7) and (6.10), and trainable scaling factors in the dynamical map, as *highly-parameterized* CGPDM. On the contrary, *lowly-parameterized* CGPDM will indicate the version that straightforwardly extends the standard GPDM structure from [96], using its same kernels, (6.6) and (6.9), and constant scaling factors. Although the adoption of kernels such (6.7) and (6.10) is a very common practice in GPR literature, to the best of our knowledge, it has not been tested before in the context of GPDM. Furthermore, also the adoption of trainable scaling factors constitutes a novelty for this kind of models.

6.2.3 Multiple sequences

It is possible to easily extend the CGPDM formulation to P multiple sequences of observations, $Y^{(1)}, \dots, Y^{(P)}$, and control inputs, $U^{(1)}, \dots, U^{(P)}$. Let the length of each sequence p, for $p = 1, \dots, P$, be equal to N_p, with $\sum_{p=1}^{P} N_p = N$. Define the latent states associated with each sequence as $X^{(1)}, \dots, X^{(P)}$. Following the notation of Section 6.2.2, define $\tilde{X}^{(1)}, \dots, \tilde{X}^{(P)}$, as the sequence of the aggregated matrices of latent states and control inputs, and $\Delta^{(1)}, \dots, \Delta^{(P)}$ as the difference matrices.

Hence, model joint likelihoods can be calculated by using the following concatenated matrices inside (6.5) and (6.8): $Y = [Y^{(1)T}| \dots |Y^{(P)T}]^T$, $X = [X^{(1)T}| \dots |X^{(P)T}]^T$, $\Delta = [\Delta^{(1)}| \dots |\Delta^{(P)}]^T$ and $\tilde{X} = [\tilde{X}^{(1)T}| \dots |\tilde{X}^{(P)T}]^T$. Note that, when dealing with multiple sequences, the number of data points in the dynamic mapping becomes $N - P$, and expression (6.8) must be adapted accordingly.

6.3 CGPDM Training and Predictions

Training the CGPDM entails using numerical optimization techniques to estimate the unknowns in the model, i.e., latent states X and the hyper-parameters θ_x, θ_y. Latent coordinates X are initialized by means of PCA [11], selecting the first d principal components of Y. A natural approach for training CGPDMs is to maximize the joint log-likelihood $\ln p(Y|X) = \ln p(Y|X) + \ln p(\Delta|\tilde{X})$ w.r.t. $\{X, \theta_x, \theta_y\}$. As regards numerical optimization, we used the L-BFGS algorithm [15].

Hence, the overall loss will be given, up to an additive constant, by

$$\mathcal{L} = \mathcal{L}_y + \mathcal{L}_x, \tag{6.11}$$

where

$$\mathcal{L}_y = \frac{D}{2}\ln|K_y(X)| + \frac{1}{2}\mathrm{tr}(K_y(X)^{-1}YW_y^2Y^T) - N\ln|W_y|,$$

$$\mathcal{L}_x = \frac{d}{2}\ln|K_x(\tilde{X})| + \frac{1}{2}\mathrm{tr}(K_x(\tilde{X})^{-1}\Delta W_x^2\Delta^T) - (N-1)\ln|W_x|.$$

In case the CGPDM is trained on multiple sequences of inputs and observations, make sure to employ the aggregated matrices defined in Section 6.2.3 when computing loss function 6.11. It is also necessary to use the factor $N - P$ instead of $N - 1$ inside the expression of $\mathcal{L}_x$.

A trained CGPDM can be used to fulfill two different purposes: (i) map a given new latent state $\boldsymbol{x}_t^*$ to the corresponding $\boldsymbol{y}_t^*$ in observation space, (ii) predict the evolution of

the latent state at the next time step $\boldsymbol{x}_{t+1}^*$, given $\boldsymbol{x}_t^*$ and a certain control action $\boldsymbol{u}_t^*$. The two processes, together, can be used to predict the sequence of observations produced by applying a series of control actions to the system.

6.3.1 Latent prediction

Given $\boldsymbol{x}_t^*$, the probability density of its corresponding observation $\boldsymbol{y}_t^*$ is $p(\boldsymbol{y}_t^*|\boldsymbol{x}_t^*, X, \theta_y) = \mathcal{N}(\mu_y(\boldsymbol{x}_t^*), v_y(\boldsymbol{x}_t^*)W_y^{-2})$, with

$$\mu_y(\boldsymbol{x}_t^*) = Y^T K_y(X)^{-1} \boldsymbol{k}_y(\boldsymbol{x}_t^*, X)$$

$$v_y(\boldsymbol{x}_t^*) = k_y(\boldsymbol{x}_t^*, \boldsymbol{x}_t^*) - \boldsymbol{k}_y(\boldsymbol{x}_t^*, X)^T K_y(X)^{-1} \boldsymbol{k}_y(\boldsymbol{x}_t^*, X),$$

where $\boldsymbol{k}_y(\boldsymbol{x}_t^*, X) = [k_y(\boldsymbol{x}_t^*, \boldsymbol{x}_1), \ldots, k_y(\boldsymbol{x}_t^*, \boldsymbol{x}_N)]^T$.

6.3.2 Dynamics prediction

Given $\boldsymbol{x}_t^*$ and $\boldsymbol{u}_t^*$, let's define $\tilde{\boldsymbol{x}}_t^* = [\boldsymbol{x}_t^{*T}, \boldsymbol{u}_t^{*T}]^T$. The probability density of the latent state at the next time step $\boldsymbol{x}_{t+1}^*$ is $p(\boldsymbol{x}_{t+1}^*|\tilde{\boldsymbol{x}}_t, X, \theta_x) = \mathcal{N}(\mu_x(\boldsymbol{x}_t^*), v_x(\boldsymbol{x}_t^*)W_x^{-2})$, with

$$\mu_x(\boldsymbol{x}_t^*) = \boldsymbol{x}_t^* + \Delta^T K_x(\tilde{X})^{-1} \boldsymbol{k}_x(\tilde{\boldsymbol{x}}_t^*, \tilde{X}),$$

$$v_x(\boldsymbol{x}_t^*) = k_x(\tilde{\boldsymbol{x}}_t^*, \tilde{\boldsymbol{x}}_t^*) - \boldsymbol{k}_x(\tilde{\boldsymbol{x}}_t^*, \tilde{X})^T K_x(\tilde{X})^{-1} \boldsymbol{k}_x(\tilde{\boldsymbol{x}}_t^*, \tilde{X}),$$

where $\boldsymbol{k}_x(\tilde{\boldsymbol{x}}_t^*, \tilde{X}) = [k_x(\tilde{\boldsymbol{x}}_t^*, \tilde{\boldsymbol{x}}_1), \ldots, k_x(\tilde{\boldsymbol{x}}_t^*, \tilde{\boldsymbol{x}}_{N-1})]^T$.

6.3.3 Trajectory prediction

Starting from an initial latent state $\boldsymbol{x}_1^*$, one can predict the evolution of the system over a desired horizon of length N_d, when subject to a given sequence of control actions $\boldsymbol{u}_1^*, \ldots, \boldsymbol{u}_{N_d-1}^*$. At each time step $t = 1, \ldots, N_d - 1$, $\boldsymbol{x}_{t+1}^*$ can be sampled from the normal distribution $p(\boldsymbol{x}_{t+1}^*|\tilde{\boldsymbol{x}}_t, X, \theta_x)$ defined in Section 6.3.2. Hence, the generated trajectory in the latent space $\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_{N_d}^*$ can be mapped into the associated predicted sequences of observations $\boldsymbol{y}_1^*, \ldots, \boldsymbol{y}_{N_d}^*$ by mean of $p(\boldsymbol{y}_t^*|\boldsymbol{x}_t^*, X, \theta_y)$, defined in Section 6.3.1.

6.4 Experiments with Simulated Cloth

As regards to the experimental application, we considered a simulated scenario, consisting of a bimanual robot that moves a piece of cloth by holding its two upper corners. The cloth is modeled as an 8×8 mesh of material points (for a total amount of $P = 64$ points).

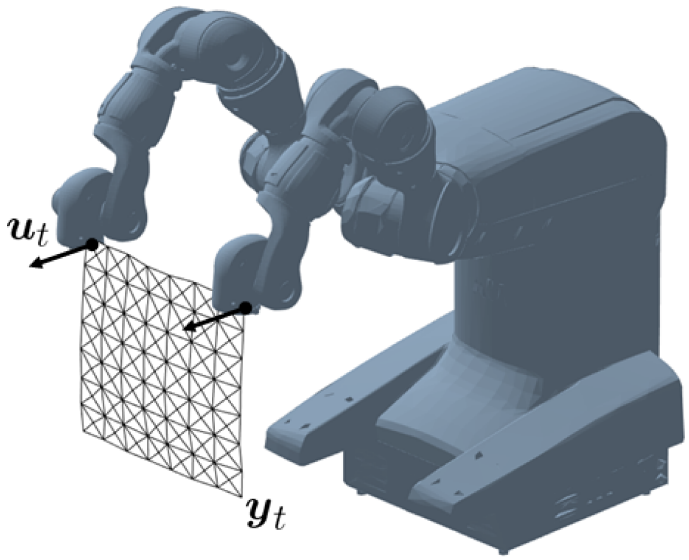


Figure 6.1: Simulated bimanual cloth manipulation setup. The cloth is positioned in its starting configuration. Observations $\boldsymbol{y}_t$ are the positions of the points in the mesh representing the cloth. Control actions $\boldsymbol{u}_t$ are the differences between consecutive end-effectors' commands.

The two points in the upper corners are assumed to be attached to the two robot's end-effectors, while the other points move following the dynamical model proposed in [22]. In this context, the observation vector is given by the Cartesian coordinates of all the points in the mesh (measured in meters); hence $\boldsymbol{y}_t \in \mathbb{R}^D$ with $D = 8 \times 8 \times 3 = 192$. We assume that the two end-effectors can be controlled exactly in the operational space. In this system, the controls acting at time step t are the differences between position commands at instant $t+1$ and t (measured in meters); so $\boldsymbol{u}_t \in \mathbb{R}^E$ with $E = 6$. The overall setup is shown in Figure 6.1. The objective of the experiment is to learn the high-dimensional cloth dynamics using CGPDM, in order to make predictions about cloth movements in response to sequences of control actions that were not seen during training. We adopted a latent space of dimension $d = 3$, resulting in a dimensionality reduction factor of $D/d = 64$. Such high-dimensional task would be unfeasible to model by standard GPR without DR. CGPDMs was implemented in Python environment, exploiting PyTorch library [72] [1]. Specifically, we would like to evaluate how much the prediction accuracy is affected by

- the number of observation sequences used for training,
- the range of the cloth movements,
- the use of *lowly-parameterized* or *highly-parameterized* CGPDM structure.

[1] Code available at `https://bitbucket.org/fabio_ama/gpdm_lib`

6.4.1 Data collection

Data were obtained by recording mesh trajectories associated with several types of cloth oscillation, obtained by applying different sequences of control actions. All the considered trajectories start from the same cloth configuration and last 3 seconds. Observations were recorded each 0.05 seconds, hence $N = 60$ total number of steps for each sequence.

Let $\boldsymbol{u}_t = \left[{}^r\delta_t^X, {}^r\delta_t^Y, {}^r\delta_t^Z, {}^l\delta_t^X, {}^l\delta_t^Y, {}^l\delta_t^Z\right]^T$, where ${}^r\delta_t^X$, ${}^r\delta_t^Y$ and ${}^r\delta_t^Z$ (${}^l\delta_t^X$, ${}^l\delta_t^Y$ and ${}^l\delta_t^Z$) indicate the displacement of the right (left) end-effector position along the three Cartesian axes, between step t and $t+1$. Specifically, we used the same commands for both end-effectors. Denoting with $\left(\delta_t^X, \delta_t^Y, \delta_t^Z\right)$ the common displacements, the applied $\boldsymbol{u}_t$ were given by,

$$\delta_t^X = 0 \tag{6.12a}$$

$$\delta_t^Y = -A\ cos(\gamma)\ cos(2\pi f_Y\ t) \tag{6.12b}$$

$$\delta_t^Z = A\ sin(\gamma)\ cos(2\pi f_Z\ t). \tag{6.12c}$$

Such controls make the end-effectors oscillate on the Y-Z plane of the operational space. The maximum displacement is regulated by A, that we set to 0.01 meters. Parameter γ can be interpreted as the inclination of $\boldsymbol{u}_1$ w.r.t. the horizontal, and it loosely defines a direction of the oscillation. f_Y and f_Z define the frequencies of the oscillations along Y and Z axes. If they are similar, the end-effectors move mostly along the direction defined by γ, if not, they swipe in a broader space.

In order to obtain a heterogeneous set of trajectories for the composition of training and test sets, we collected several movements obtained by choosing in a random fashion the control parameters γ, f_Y and f_Z. Angles γ were uniformly sampled from a variable range $[-\frac{R}{2}, \frac{R}{2}]$ (deg), ; in the following, we indicate this range with the amplitude of its angular area, R (deg). Instead, frequencies f_Y and f_Z were uniformly sampled inside the fixed interval [0.3, 0.6] (Hz).

We considered four movement ranges of increasing size, namely $R \in \{30, 60, 90, 120\}$ (see Figure 6.2), and collected a specific data-set $\mathcal{D}_R$ associated with each range. Every set contains 50 cloth trajectories obtained by applying control actions of the form (6.12) with 50 different random choices for parameters γ, f_Y and f_Z. From each collection $\mathcal{D}_R$, 10 trajectories were extracted and used as test sets $\mathcal{D}_R^{test}$ for the corresponding movement range, while several training sets $\mathcal{D}_R^{train}$ were built by randomly picking from the remaining sequences.

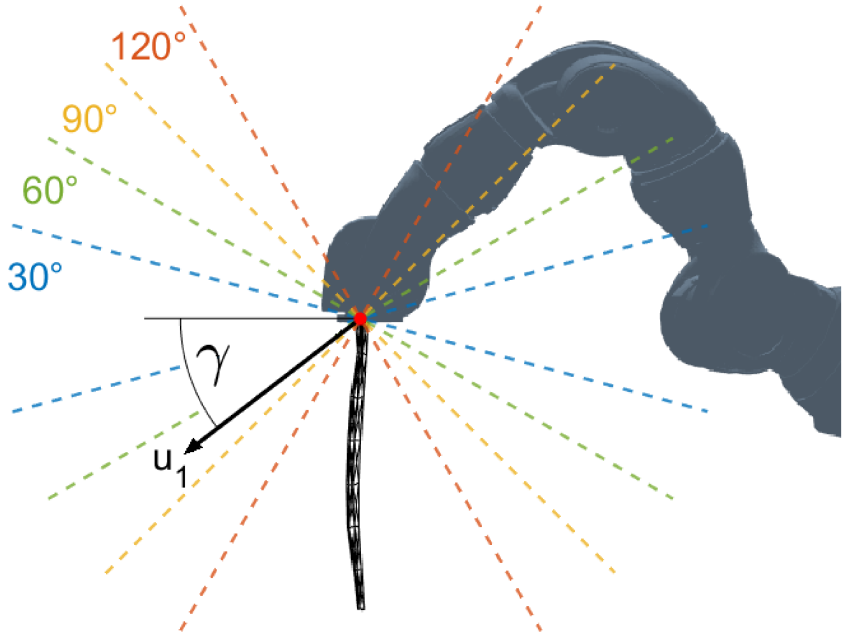


Figure 6.2: Representation of the movement ranges within which control parameter γ was sampled during data collection.

6.4.2 Model training

The objective of the experiment is to evaluate CGPDM prediction accuracy in different movement ranges, and for different amounts of training data. Also, we want to observe if the use of *lowly-parameterized* or *highly-parameterized* CGPDMs yield a substantial difference in terms of prediction accuracy.

Consequently, for each considered movement range R, we trained different *highly-parameterized* CGPDMs employing an increasing number of sequences randomly picked from $\mathcal{D}_R^{train}$. Specifically, we used 10 different combinations of 5, 10, 15 and 20 sequences for each range. In this way, we were able to reduce the dependencies on the specific training trajectories employed, and to average prediction accuracy over different possible sets of training data. For each such model, a *lowly-parameterized* CGPDM was trained on exactly the same data.

6.4.3 Results

We used each learned CGPDM to predict the cloth movements when subject to the test control actions relative to the associated $\mathcal{D}_R^{test}$, for $R \in \{30, 60, 90, 120\}$. Let $\boldsymbol{y}_t^{(R,k)}$ and $\boldsymbol{u}_t^{(R,k)}$ denote the observation and control action, at time step t, of the k-th test trajectory in $\mathcal{D}_R^{test}$, with $k = 1, \ldots, 10$.

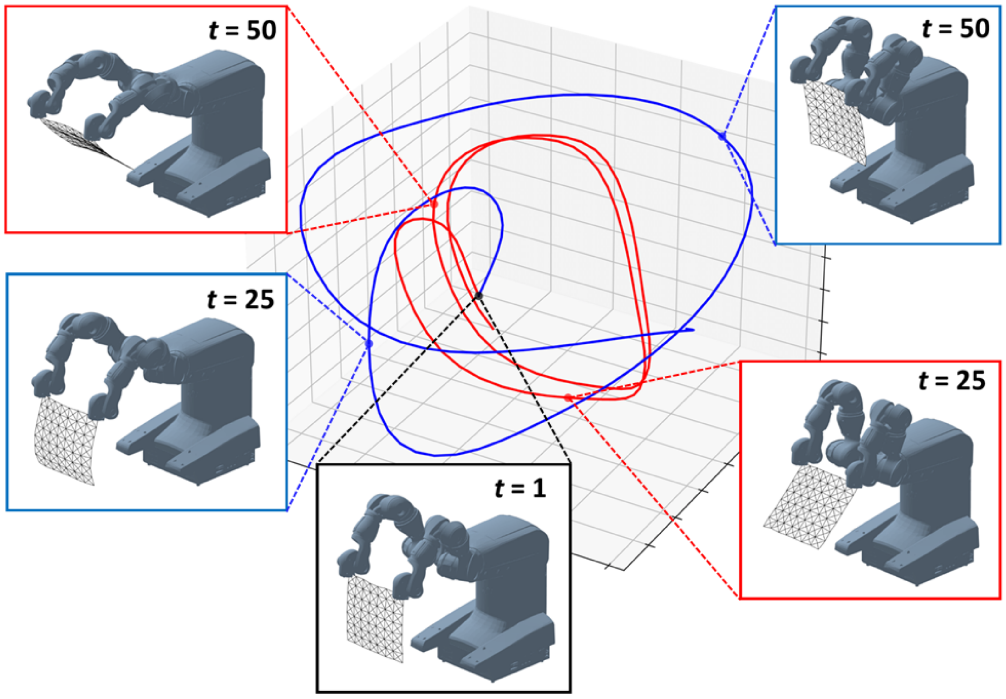


Figure 6.3: The same CGPDM predicts two distinct latent state trajectories in response to two different sequences of unseen control actions. Each point in the latent space has associated a particular configuration of the modeled cloth.

For every considered range R, one can follow the procedure described in Section 6.3.3 and employ the trained CGPDMs to predict the trajectories resulting from the application of control action sequences $\{\boldsymbol{u}_t^{(R,k)}\}_{t=1}^{N-1}$, for $k = 1, \ldots, 10$. Let $\boldsymbol{x}_t^{*(R,k)}$ be the predicted latent state at time t, and $\boldsymbol{y}_t^{*(R,k)}$ the corresponding predicted observation. As an example, in Figure 6.3, we show the latent space trajectories predicted by one of the trained CGPDM subject to two different sequences of control actions. Note how the two latent trajectories, starting from the same initial state, are driven by their respective control actions into different regions of the latent space, where different cloth poses are reconstructed.

There is a degree of freedom in the choice of the initial latent state. Since in the GPDM formulation, an explicit map from observation to latent space is not given; there is no principled way to choose $\boldsymbol{x}_1^{*(R,k)}$ given an unseen $\boldsymbol{y}_1^{*(R,k)}$. But, in the context of this experiment, all the cloth movements adopted for training and test started from the same configuration. Hence, we could select $\boldsymbol{x}_1^{*(R,k)}$ equal to the latent state associated with the first observation of the training trajectories.

By visualization of the predicted movements [2], we can state that CGPDMs, trained

[2] A video of the reconstructed simulated cloth motions is available at `https://youtu.be/vUO_3nYgMeg`

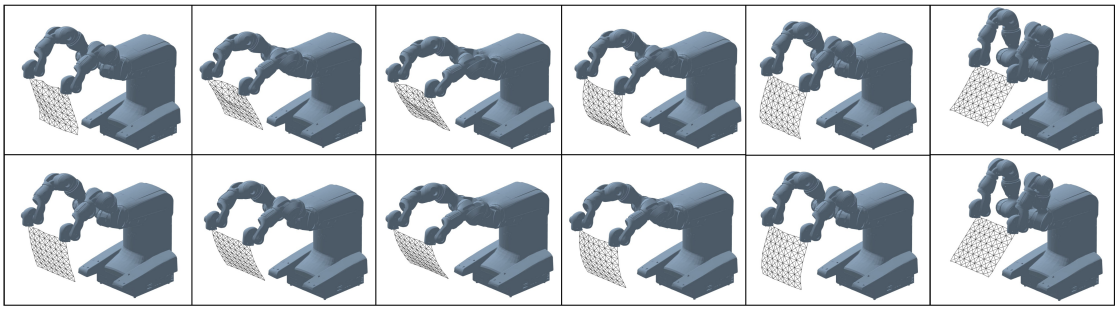


Figure 6.4: (Top) Frames recorded from a simulated cloth movement. (Bottom) Corresponding predictions obtained by CGPDM.

with a sufficient amount of data (10, 15 and 20 sequences in this example), are able to capture the cloth dynamics of oscillations along axes Y and Z. Such models obtained satisfying results in a variety of movement ranges. In fact, for smaller movement ranges ($R = 30$ or $R = 60$), the reconstructed trajectories of the mesh of points appear very similar to the true ones. On the other hand, for wider ranges ($R = 90$ or $R = 120$), discrepancies between true and predicted points begin to be more evident, but the CGPDMs are still able to predict the overall movement of the piece of cloth quite faithfully. In Figure 6.4 we report a series of video frames showing the original cloth movement and the relative prediction.

Let us indicate the Cartesian coordinates of the p-th mesh point at time step t as $[x_t^p, y_t^p, z_t^p]$. Then, for each range R, it is possible to rewrite the true observation from the k-th test trajectory at time step t, as $\boldsymbol{y}_t^{(R,k)} = [x_t^1, \ldots, x_t^P, y_t^1, \ldots, y_t^P, z_t^1, \ldots, z_t^P]^T$. Analogously, $\boldsymbol{y}_t^{*(R,k)}$ can be rewritten as $\boldsymbol{y}_t^{*(R,k)} = [x_t^{1*}, \ldots, x_t^{P*}, y_t^{1*}, \ldots, y_t^{P*}, z_t^{1*}, \ldots, z_t^{P*}]^T$. Following this notation, it is possible to define, for each angular range $R \in \{30, 60, 90, 120\}$ and every test trajectory $k = 1, \ldots, 10$, the prediction error $e^{(R,k)}$ as the average distances between the true mesh points and the predicted ones, whose expression is given by

$$e^{(R,k)} = \frac{1}{N} \sum_{t=1}^{N} \frac{1}{P} \sum_{p=1}^{P} \sqrt{(x_t^p - x_t^{p*})^2 + (y_t^p - y_t^{p*})^2 + (z_t^p - z_t^{p*})^2}. \tag{6.13}$$

In Figure 6.5, we report, for all the movement ranges, the mean prediction errors (6.13) obtained in the test sets by CGPDMs, averaging the results over the different training sets adopted (as indicated in Section 6.4.2, we repeated all the experiments 10 times, using a $\mathcal{D}_R^{train}$ of random composition every turn). Results are indicated with mean and 95% confidence interval.

As expected, CGPDMs trained with only 5 sequences show higher errors than the models trained employing more data. But the resulting errors do not always diminish with

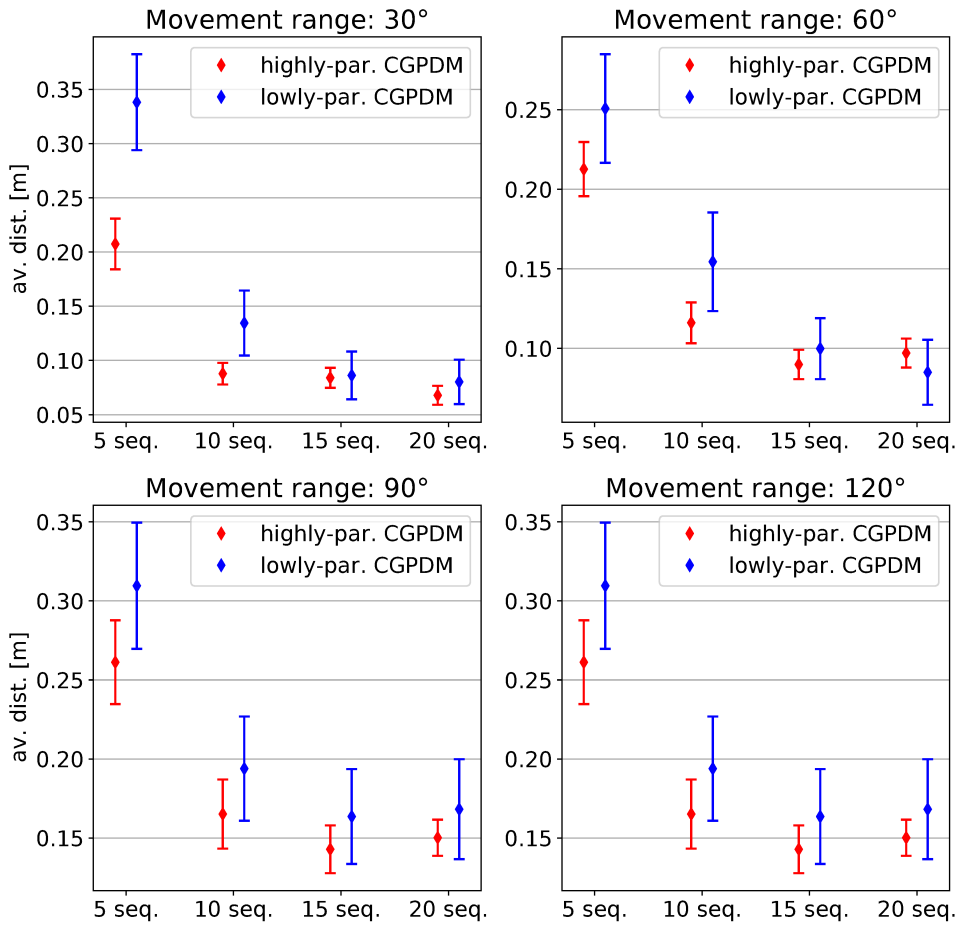
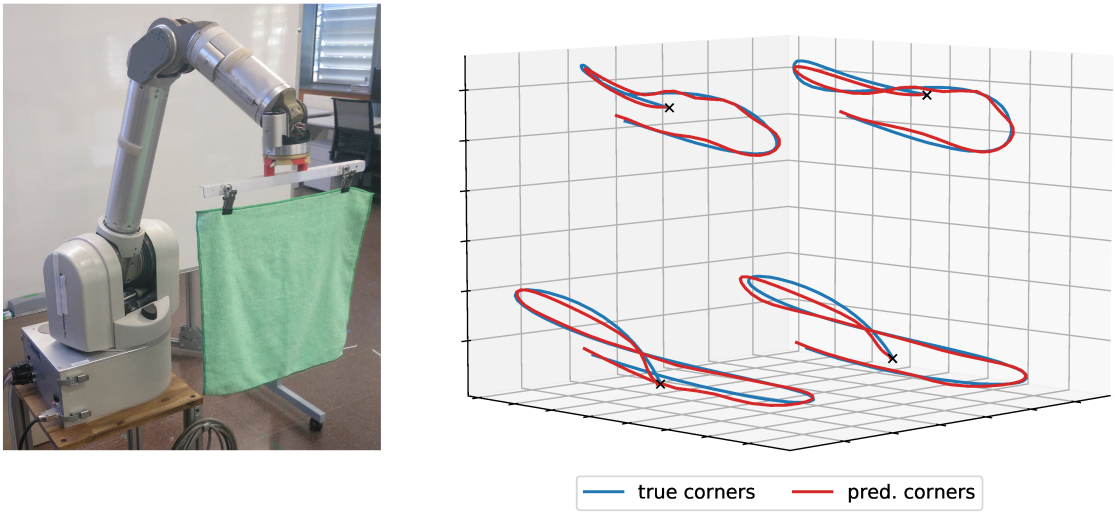


Figure 6.5: Average point-to-point distances (with 95% C.I.) obtained by different CGPDM setups in the considered movement ranges.

the increase in the amount of training trajectories. In fact, in all the considered movement ranges, accuracy does not considerably change passing from 15 training sequences to 20. Moreover, the proposed *highly-parameterized* CGPDM structure seems able to improve accuracy and consistency of the results in the majority of cases. This effect is clearer in a low-data regime, with models trained on 5 or 10 sequences.

6.5 Experiments with Real Cloth

Finally, we tested CGPDM on data collected in a real cloth manipulation experiment. For this purpose, we used a Barrett WAM Arm, whose end-effector consists of a coat rack that can firmly grip a piece of cloth from its corners. The overall setup is depicted in Fig. 6.6a. The task considered is totally analogous to the one simulated in Sec. 6.4, despite



(a) Real experimental setup.

(b) True and predicted corner trajectories for a real cloth movement.

Figure 6.6: Picture of the real experimental setup and modeling results.

having here only a single robot involved. We controlled robot's end-effector in position, recording the resulting movement of the cloth through a motion capture system. We combined object detection, image and point cloud processing for segmenting cloth-like objects from point clouds, based on [12], [80] and [98] [3]

6.5.1 Data Collection

As before, we captured the cloth as a 8×8 mesh of points, whose coordinates constitute the observation vector $\boldsymbol{y}_t \in \mathbb{R}^D$ with $D = 192$. As control actions, we considered the difference between consecutive position commands (with fixed orientation); hence $\boldsymbol{u}_t \in \mathbb{R}^E$ with $E = 3$. Robot was controlled at 100 Hz, with commands chosen following (6.12).

In this case, f_Y and f_Z have been uniformly sampled in [0.2, 0.5] (Hz) and A was reduced to 0.004 meters, due to the higher control frequency. On the other hand, the motion capture system could work only at lower rates, with no guarantees on the sampling interval. Thus, it was necessary to post-process the data to make them ready for modeling. Firstly, motion capture data were smoothed by a moving average filter. Then we interpolated the positions of both the end-effector and the cloth mesh, to obtain two synchronized sequences of observations and control actions, sampled at 20 Hz. We collected 20 different 3 second long trajectories for each of the two ranges, $R = 30$ and $R = 60$ (40 sequences in total).

[3] Code available at `https://github.com/MiguelARD/cloth_point_cloud_segmentation`

6.5.2 Model training & Results

We trained two CGPDMs model, one for $R = 30$ and the other for $R = 60$. We adopted the *highly-parameterized* structure, as it performed better in the simulated experiment. For each R, the CGPDM was trained using ten trajectories, while the others were set aside for testing the trained model. As before, we set latent space dimension to $d = 3$.

We tested the two trained CGPDM following the same procedure that was adopted in Sec. 6.4.3. The two models were used to predict the cloth movements obtained in response to the control actions of each test trajectory [4]. The prediction error was defined as in (6.13), and, in range $R = 30$, it was 0.012 ± 0.005, while, in range $R = 60$, it was 0.015 ± 0.006. These results are slightly inferior to those obtained in the simulated experiment (Fig. 6.5). Nevertheless, CGPDMs seem able to cope with the high noise that afflicts the real experimental setup and still capture the dynamics of the cloth. In Fig. 6.6b, we provide a visual representation of the cloth movements, by representing the true and predicted trajectories of the four corners, for one of the considered test cases.

6.6 Conclusion

We presented an extension of the GPDM framework able to incorporate control actions in its definition, we call it *Controlled Gaussian Process Dynamical Model* (CGPDM). Such framework can model complex high-dimensional dynamics governed by control actions, by projecting observation space into a latent space of lower dimensionality, where dynamical relations are easier to infer. Alongside, we discussed two different possible implementation of CGPDMs. The first, called *lowly-parameterized* CGPDM, is a straightforward extension of the original GPDM structure. The latter, indicated as *highly-parameterized* CGPDM, employs a richer structure for both latent and dynamical maps. CGPDM was tested in both a simulated and real scenario of robotic cloth manipulation. Specifically, we considered a robot moving a piece of cloth by holding it from its two upper corners. The cloth was represented as an 8×8 mesh of points, resulting in an observation vector that comprises all their Cartesian coordinates, for a total dimension of $D = 192$. In both scenarios, we projected the cloth movement into a latent space of dimension $d = 3$. On the other hand, we considered as control actions the differences between end-effector positions at consecutive time steps. The simulated environment was exploited to train and test several realizations of CGPDM on a sufficiently varied data-set of trajectories. The proposed model was able to learn the complex high-dimensional dynamics of the cloth oscillations. Furthermore, the *highly-parameterized* CGPDM structure achieved the

[4] A video of the reconstructed real cloth motions is available at `https://youtu.be/vUO_3nYgMeg`

best prediction accuracy. Finally, this result was validated on a real cloth manipulation scenario, where data were collected through a motion capture system. CGPDM proved capable of effectively handle noisy real data.

Future work involves the application of CGPDM to tasks involving contacts between the cloth and other objects. CGPDMs may also be employed in a model-based reinforcement learning framework that relies on GPs to handle the modeling of high-dimensional dynamics, that could be unfeasible to approach without applying dimensionality-reduction techniques. Finally, the proposed CGPDM formulation could be extended through the introduction of back constraints [56] to preserve local distances and obtain an explicit formulation of the mapping from the observation to latent space.

7 Conclusions

In this manuscript, we presented different algorithms based on Gaussian Processes for applications to Model-Based Reinforcement Learning, data-driven torque control for robots, and the modeling of high-dimensional dynamics.

Specifically, Chapter 2 introduced a novel Model-Based Reinforcement Learning algorithm, called MC-PILCO. The proposed method exploits Gaussian Process Regression to derive a probabilistic predictive model of the controlled system. From this model, we can estimate the expected cumulative cost from which it is possible to derive, through Monte Carlo sampling, the gradient needed to update the parameters of the policy. The long-term predictions are obtained simulating numerous state particles. In this way, differently for other algorithms, we do not need to make any approximations on long-term evolution, neither we pose restrictions on the choice of the kernel. MC-PILCO managed to outperform other state-of-the-art Model-Based Reinforcement Learning algorithms in the cart-pole benchmark. Also, we applied it to learn a trajectory tracking controller for a simulated UR5 robot. MC-PILCO achieved unprecedented levels of data efficiency in all the conducted experiments.

Thanks to the flexibility guaranteed by Monte Carlo policy gradient estimation, we were able to easily adapt the MC-PILCO framework to the context of partially measurable systems, i.e., systems whose state cannot be measured directly in all its components. We argue that this is a common situation in many real-world applications. For instance,

when dealing with mechanical systems, we must reconstruct joint velocities from position measurements. In Chapter 3 we proposed a modified version of the original MC-PILCO algorithm, designed precisely to deal with partial state measurability. We called this method MC-PILCO4PMS, and it can correctly incorporate state estimators inside model and policy learning. This approach was able to learn from scratch how to control two different real mechanical systems, namely a Furuta pendulum and a ball-and-plate.

In Chapter 4, we modified our framework to work only with the history of position measurements. In this way, it is possible to avoid performing velocity estimation. In fact, the tuning of effective filters and estimators might be a tedious and time-consuming task, especially for those systems that are affected by high measurement noise. The last of our Model-Based Reinforcement Learning algorithms is called Derivative-Free MC-PILCO (DF-MC-PILCO). We derived it from standard MC-PILCO by considering, instead of the normal state representation, the history of past position measurements and applied control actions. DF-MC-PILCO performed as well as MC-PILCO4PMS, both in simulation and in the two real-world benchmarks, requiring similar amount of experience for finding a solution.

In Chapter 5 we employed Gaussian Process Regression to estimate robot inverse dynamics for control purposes. Indeed, a well-known strategy in robotics is to use inverse dynamics to perform feedback linearization control. Our method aims at using purely data-driven models in the same way. In general, Gaussian Processes equipped with standard Squared Exponential kernels show poor generalization properties outside of the training data. This is a critical issue if we want to employ such models inside the feedback linearization control loop. Still, we managed to circumvent this problem through two alternative strategies. The first uses a particular kernel function that was designed specifically for inverse dynamics learning. The second extracts estimate all the different dynamics components needed for feedback linearization from the data-driven model. Experimental results were obtained on a simulated KUKA LBR iiwa robot.

Finally, Chapter 6 describes how to exploit Gaussian Processes to model the high-dimensional dynamics characterizing cloth manipulation tasks. In particular, we proposed a novel model, called CGPDM, that exploits Gaussian Process Regression in two ways: (i) to build a latent map that can project highly-dimensional observations into a latent space of tractable dimension; (ii) to learn a latent dynamical function able to capture the effects that control actions have on the general behaviour of the time series. This work is based on previous latent models based on Gaussian Processes, namely GPLVM and GPDM, but we introduced for the first time control actions in such formulation. Without taking into account control, previous models could only generate synthetic data,

similar to the one observed during training. CGPDMs, on the other hand, can predict the effects that new control actions have on the high-dimensional system. The proposed method was used to model the movements of a piece of cloth handled by a robot. We modeled the dynamics of a mesh of material points, commonly used as a representation of cloth configuration. CGPDM was tested with both simulated and real data, proving able to generalize over previously unseen control actions, in the two considered scenarios.

A

Appendix

A.1 Semi-Parametrical Kernel for the Cart-pole System

Let us consider the cart-pole system adopted as benchmark in Section 2.5. It consists of a cart with mass M and an attached pendulum with mass m and length L, which swings freely in the plane. The pendulum angle θ is measured anti-clockwise from hanging down. The cart can move horizontally with an applied external force u. The position of the cart along the track is denoted by p. We refer with b to the coefficient of friction between the cart and the ground. The basis functions of the Semi-Parametrical (SP) kernel must be derived from the true equations of motion. For this purpose, we make use of the Lagrangian approach.

The coordinates of the pendulum midpoint (x, y) are given by

$$x = p + \frac{1}{2}L\ sin(\theta)$$
$$y = -\frac{1}{2}L\ cos(\theta)$$

while its squared velocity v^2 can be obtained as

$$v^2 = \dot{x}^2 + \dot{y}^2 = \dot{p}^2 + \frac{1}{4}L^2\dot{\theta}^2 + L\dot{p}\dot{\theta}\ cos(\theta).$$

On the other hand, the squared velocity of the cartpole midpoint is simply $\dot{p}^2$. Now, recalling the expression of the moment of inertia of a pendulum around its midpoint $I = \frac{1}{12}mL^2$, the total kinetic energy of the system T can be calculated as

$$T = \frac{1}{2}M\dot{p}^2 + \frac{1}{2}mv^2 + \frac{1}{2}I\dot{\theta}^2.$$

We are also interested in the expression of the potential energy, that is defined as $V = mgy$, where g is the gravity acceleration. Finally, the Lagrangian of the system L can be obtained as

$$L = T - V = \frac{1}{2}(M+m)\dot{p}^2 + \frac{1}{6}mL^2\dot{\theta}^2 + \frac{1}{2}mL(\dot{x}\dot{\theta} + g)\ cos(\theta).$$

Now, rename the generalized coordinates of the cart-pole system as $q_1 = p$ and $q_2 = \theta$ and define $\tau_1 = u$ and $\tau_2 = 0$. The equations of motion can be derived from

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = \tau_i \qquad \textbf{for} \quad i = 1, 2.$$

Leaving out the explicit calculations, we obtain the following final expressions,

$$\begin{aligned}(M+m)\ddot{p} + \frac{1}{2}mL\ddot{\theta}\ cos(\theta) - \frac{1}{2}mL\dot{\theta}^2\ sin(\theta) &= u - b\dot{p} \\ 2L\ddot{\theta} + 3\dot{p}\ cos(\theta) + 3g\ sin(\theta) &= 0\end{aligned}$$

Collecting the state variables $\boldsymbol{x} = [p, \dot{p}, \theta, \dot{\theta}]^T$, the equations of motions can be conveniently expressed as four coupled ordinary differential equations,

$$\begin{cases} \ddot{p} = \dfrac{2mL\dot{\theta}^2\ sin(\theta) + 3mg\ sin(\theta)cos(\theta) + 4u - 4b\dot{x}}{4(M+m) - 3m\ cos(\theta)^2} \\ \ddot{\theta} = \dfrac{-3mL\dot{\theta}^2\ sin(\theta)cos(\theta) - 6(M+m)g\ sin(\theta) - 6(u - b\dot{x})cos(\theta)}{4L(M+m) - 3Lm\ cos(\theta)^2} \end{cases}$$

Now, from these equations we can define the basis functions for the Physically Inspired component of the SP kernel (details about this kind of kernel are given in Sec. 2.3.2).

Let us indicate with $\phi_{\dot{p}}(\boldsymbol{x}, u)$ and $\phi_{\dot{\theta}}(\boldsymbol{x}, u)$, respectively, the basis functions for the Gaussian Processes that predict velocity changes of cart and pole's angle. Those basis functions can be obtained by isolating, in each ordinary differential equation, all the different state-dependent components that are linearly related. In particular, we obtain

$$\begin{aligned}\phi_{\dot{p}}(\boldsymbol{x}, u) &= [\dot{\theta}^2\ sin(\theta), sin(\theta)cos(\theta), u, \dot{x}] \\ \phi_{\dot{\theta}}(\boldsymbol{x}, u) &= [\dot{\theta}^2\ sin(\theta)cos(\theta), sin(\theta), u\ cos(\theta), \dot{x}\ cos(\theta)]\end{aligned}.$$

References

[1] **Abbeel P., Coates A., Quigley M., and Ng A.** An application of reinforcement learning to aerobatic helicopter flight. In **Schölkopf B., Platt J., and Hoffman T.**, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2007. URL `https://proceedings.neurips.cc/paper/2006/file/98c39996bf1543e974747a2549b3107c-Paper.pdf`.

[2] **Alberto N. T., Mistry M., and Stulp F.** Computed torque control with variable gains through gaussian process regression. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 212–217, 2014.

[3] **Atkeson C. G. and Santamaria J. C.** A comparison of direct and model-based reinforcement learning. In *Proceedings of international conference on robotics and automation*, volume 4, pages 3557–3564. IEEE, 1997.

[4] **Baldassi C., Malatesta E. M., and Zecchina R.** Properties of the geometry of solutions and capacity of multilayer neural networks with rectified linear unit activations. *Phys. Rev. Lett.*, 123:170602, Oct 2019. URL `https://link.aps.org/doi/10.1103/PhysRevLett.123.170602`.

[5] **Baldassi C., Pittorino F., and Zecchina R.** Shaping the learning landscape in neural networks around wide flat minima. *Proceedings of the National Academy of Sciences*, 117(1):161–170, 2020. ISSN 0027-8424. URL `https://www.pnas.org/content/117/1/161`.

[6] **Baraff D. and Witkin A.** Large steps in cloth simulation. *Association for Computing Machinery*, pages 43–54, 1998.

[7] **Baraff D. and Witkin A.** Dexterous manipulation of cloth. *Computer Graphics Forum*, 35(2):523–532, 2016.

[8] **Beckers T., Umlauft J., Kulic D., and Hirche S.** Stable gaussian process based tracking control of lagrangian systems. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 5180–5185, 2017.

[9] **Berkenkamp F., Turchetta M., Schoellig A., and Krause A.** Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*, pages 908–918, 2017.

[10] **Bersch C., Pitzer B., and Kammel S.** Bimanual robotic cloth manipulation for laundry folding. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1413–1419, 2011.

[11] **Bishop C. M.** *Pattern recognition and machine learning*. Springer, 2006.

[12] **Bochkovskiy A., Wang C.-Y., and Liao H.-Y. M.** Yolov4: Optimal speed and accuracy of object detection, 2020.

[13] **Borràs J., Alenyà G., and Torras C.** A grasping-centered analysis for cloth manipulation. *IEEE Transactions on Robotics*, PP:1–13, 05 2020.

[14] **Bottou L.** Large-scale machine learning with stochastic gradient descent. In *in COMPSTAT*, 2010.

[15] **Byrd R. H., Lu P., Nocedal J., and Zhu C.** A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5): 1190–1208, 1995.

[16] **Caccavale F., Siciliano B., and Villani L.** Robot impedance control with nondiagonal stiffness. *IEEE Transactions on Automatic Control*, 44(10):1943–1946, 1999.

[17] **Caflisch R. E. and others** . Monte carlo and quasi-monte carlo methods. *Acta numerica*, 1998:1–49, 1998.

[18] **Cazzolato B. S. and Prime Z.** On the dynamics of the furuta pendulum. *Journal of Control Science and Engineering*, 2011, 2011.

[19] **Chatzilygeroudis K., Rama R., Kaushik R., Goepp D., Vassiliades V., and Mouret J.-B.** Black-box data-efficient policy search for robotics. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 51–58. IEEE, 2017.

[20] **Chua K., Calandra R., McAllister R., and Levine S.** Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.

[21] **Colomé A. and Torras C.** Dimensionality reduction for dynamic movement primitives and application to bimanual manipulation of clothes. *IEEE Transactions on Robotics*, 34(3):602–615, 2018.

[22] **Coltraro F., Amorós J., Alberich-Carramiñana M., and Torras C.** An inextensible model for robotic simulations of textiles, 2021.

[23] **Coumans E. and Bai Y.** Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019.

[24] **Craig J. J.** *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Longman Publishing Co., Inc., USA, 2nd edition, 1989. ISBN 0201095289.

[25] **Csató L. and Opper M.** Sparse on-line gaussian processes. *Neural Comput.*, 14(3):641–668, March 2002. ISSN 0899-7667. URL `https://doi.org/10.1162/089976602317250933`.

[26] **Cutler M. and How J. P.** Efficient reinforcement learning for robots using informative simulated priors. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2605–2612. IEEE, 2015.

[27] **Dalla Libera A., Romeres D., Jha D. K., Yerazunis B., and Nikovski D.** Model-based reinforcement learning for physical systems without velocity and acceleration measurements. *IEEE Robotics and Automation Letters*, 5(2):3548–3555, 2020.

[28] **Dalla Libera A., Romeres D., Jha D. K., Yerazunis B., and Nikovski D.** Model-based reinforcement learning for physical systems without velocity and acceleration measurements. *IEEE Robotics and Automation Letters*, 5(2):3548–3555, 2020.

[29] **Dalla Libera A., Tosello E., Pillonetto G., Ghidoni S., and Carli R.** Proprioceptive robot collision detection through gaussian process regression. In *2019 American Control Conference (ACC)*, pages 19–24, 2019.

[30] **Dalla Libera A., Carli R., and Pillonetto G.** A novel multiplicative polynomial kernel for volterra series identification. *IFAC-PapersOnLine (to appear)*, 2020. URL `http://arxiv.org/abs/1905.07960`. 21th IFAC World Congress.

[31] **De Luca A. and Ferrajoli L.** A modified newton-euler method for dynamic computations in robot fault detection and control. In *2009 IEEE International Conference on Robotics and Automation*, pages 3359–3364, 2009.

[32] **de Luca A. and Lucibello P.** A general algorithm for dynamic feedback linearization of robots with elastic joints. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, volume 1, pages 504–510 vol.1, 1998.

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

[34] **Deisenroth M. P., Calandra R., Seyfarth A., and Peters J.** Toward fast policy search for learning legged locomotion. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1787–1792. IEEE, 2012.

[35] **Deisenroth M. P., Fox D., and Rasmussen C. E.** Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):408–423, 2013.

[36] **Deisenroth M. P., Rasmussen C. E., and Fox D.** Learning to control a low-cost manipulator using data-efficient reinforcement learning. *Robotics: Science and Systems VII*, pages 57–64, 2011.

[37] **Delgado-Guerrero J. A., Colomé A., and Torras C.** Contextual policy search for micro-data robot motion learning through covariate gaussian process latent variable models. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5511–5517, 2020.

[38] **Einicke G. A.** Optimal and robust noncausal filter formulations. *IEEE Transactions on Signal Processing*, 54(3):1069–1077, 2006.

[39] **Gal Y. and Ghahramani Z.** Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1050–1059. JMLR.org, 2016.

[40] **Gal Y., McAllister R., and Rasmussen C. E.** Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, volume 4, page 34, 2016.

[41] **Garcia-Camacho I., Lippi M., Welle M. C., Yin H., Antonova R., Varava A., Borras J., Torras C., Marino A., Alenyà G., and Kragic D.** Benchmarking bimanual cloth manipulation. *IEEE Robotics and Automation Letters*, 5 (2):1111–1118, 2020.

[42] **Glorot X., Bordes A., and Bengio Y.** Deep sparse rectifier neural networks. In **Gordon G., Dunson D., and Dudík M.**, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL `http://proceedings.mlr.press/v15/glorot11a.html`.

[43] **Haddadin S., De Luca A., and Albu-Schäffer A.** Robot collisions: A survey on detection, isolation, and identification. *IEEE Transactions on Robotics*, 33(6): 1292–1312, 2017.

[44] **Hogan N.** Impedance control: An approach to manipulation. In *1984 American Control Conference*, pages 304–313, 1984.

[45] **Kalman R. E.** A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960. ISSN 0021-9223. URL `https://doi.org/10.1115/1.3662552`.

[46] **Kingma D. P. and Ba J.** Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[47] **Kingma D. P. and Welling M.** Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[48] **Kober J. and Peters J.** Policy search for motor primitives in robotics. In **Koller D., Schuurmans D., Bengio Y., and Bottou L.**, editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2009. URL `https://proceedings.neurips.cc/paper/2008/file/7647966b7343c29048673252e490f736-Paper.pdf`.

[49] **Kocijan J., Girard A., Banko B., and Murray-Smith R.** Dynamic systems identification with gaussian processes. *Mathematical and Computer Modelling of Dynamical Systems*, 11(4):411–424, 2005. URL `https://doi.org/10.1080/13873950500068567`.

[50] **Koganti N., Ngeo J. G., Tomoya T., Ikeda K., and Shibata T.** Cloth dynamics modeling in latent spaces and its application to robotic clothing assistance. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3464–3469. IEEE, 2015.

[51] **Koganti N., Shibata T., Tamei T., and Ikeda K.** Data-efficient learning of robotic clothing assistance using bayesian gaussian process latent variable model. *Advanced Robotics*, 33:1–15, 04 2019.

[52] **Koganti N., Tamei T., Ikeda K., and Shibata T.** Bayesian nonparametric learning of cloth models for real-time state estimation. *IEEE Transactions on Robotics*, 33(4):916–931, 2017.

[53] **Kuss M. and Rasmussen C. E.** Gaussian processes in reinforcement learning. In *Advances in neural information processing systems*, pages 751–758, 2004.

[54] **Lakshmanan K., Sachdev A., Xie Z., Berenson D., Goldberg K., and Abbeel P.** A constraint-aware motion planning algorithm for robotic folding of clothes. In *Experimental Robotics*, pages 547–562. Springer, 2013.

[55] **Lawrence N. and Hyvärinen A.** Probabilistic non-linear principal component analysis with gaussian process latent variable models. *Journal of machine learning research*, 6(11), 2005.

[56] **Lawrence N. D. and Quinonero-Candela J.** Local distance preservation in the gp-lvm through back constraints. In *Proceedings of the 23rd international conference on Machine learning*, pages 513–520, 2006.

[57] **LeCun Y., Bengio Y., and Hinton G.** Deep learning. *Nature*, 521:436–44, 05 2015.

[58] **Levine S., Finn C., Darrell T., and Abbeel P.** End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17(1):1334–1373, jan 2016. ISSN 1532-4435.

[59] **Li P. and Chen S.** A review on gaussian process latent variable models. In *CAAI Transactions on Intelligence Technology,*, volume 1, pages 366–376, 2016.

[60] **Libera A. D. and Carli R.** A data-efficient geometrically inspired polynomial kernel for robot inverse dynamic. *IEEE Robotics and Automation Letters*, 5(1): 24–31, 2020.

[61] **Libera A. D., Carli R., and Pillonetto G.** A novel multiplicative polynomial kernel for volterra series identification. *arXiv preprint arXiv:1905.07960*, 2019.

[62] **MacKay D. J.** *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992.

[63] **McHutchon A. J. and others** . *Nonlinear modelling and control using Gaussian processes*. PhD thesis, Citeseer, 2015.

[64] **Miller S., van den Berg J., Fritz M., Darrell T., Goldberg K., and Abbeel P.** A geometric approach to robotic laundry folding. *The International Journal of Robotics Research*, 31(2):249–267, 2012.

[65] **Neal R. M.** *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[66] **Nealen A., Muller M., Keiser R., Boxerman E., and Carlson M.** Physically based deformable models in computer graphics. *Computer Graphics Forum*, 25: 809–836, 2006.

[67] **Ng A. Y. and Jordan M. I.** Pegasus: A policy search method for large mdps and pomdps. *arXiv preprint arXiv:1301.3878*, 2013.

[68] **Nguyen-Tuong D. and Peters J.** Using model knowledge for learning inverse dynamics. In *2010 IEEE International Conference on Robotics and Automation*, pages 2677–2682, 2010.

[69] **Nguyen-Tuong D., Seeger M., and Peters J.** Computed torque control with nonparametric regression models. In *2008 American Control Conference*, pages 212–217, 2008.

[70] **Parmas P., Rasmussen C. E., Peters J., and Doya K.** Pipps: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*, pages 4065–4074, 2018.

[71] **Paszke A., Gross S., Chintala S., Chanan G., Yang E., DeVito Z., Lin Z., Desmaison A., Antiga L., and Lerer A.** Automatic differentiation in pytorch. In *Proceedings of Neural Information Processing Systems*, 2017.

[72] **Paszke A., Gross S., Massa F., Lerer A., Bradbury J., Chanan G., Killeen T., Lin Z., Gimelshein N., Antiga L., Desmaison A., Kopf A., Yang E., DeVito Z., Raison M., Tejani A., Chilamkurthy S., Steiner B., Fang L., Bai J., and Chintala S.** Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems 32*, pages 8024–8035, 2019.

[73] **Quinonero Candela J. and Rasmussen C.** A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6: 1935–1959, December 2005.

[74] **Rasmussen C. E. and Williams C. K. I.** *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

[75] **Rezaei-Shoshtari S., Meger D., and Sharf I.** Cascaded gaussian processes for data-efficient robot dynamics learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6871–6877, 2019.

[76] **Rezende D. J., Mohamed S., and Wierstra D.** Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

[77] **Romeres D., Zorzi M., Camoriano R., Traversaro S., and Chiuso A.** Derivative-free online learning of inverse dynamics models. *IEEE Transactions on Control Systems Technology*, 28(3):816–830, 2020.

[78] **Romeres D., Jha D. K., DallaLibera A., Yerazunis B., and Nikovski D.** Semiparametrical gaussian processes learning of forward dynamical models for navigating in a circular maze. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3195–3202. IEEE, 2019.

[79] **Romeres D., Zorzi M., Camoriano R., and Chiuso A.** Online semi-parametric learning for inverse dynamics modeling. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 2945–2950. IEEE, 2016.

[80] **Rother C., Kolmogorov V., and Blake A.** "grabcut": Interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, August 2004. ISSN 0730-0301. URL `https://doi.org/10.1145/1015706.1015720`.

[81] **Rumelhart D. E., Hinton G. E., and Williams R. J.** *Learning Representations by Back-Propagating Errors*, page 696–699. MIT Press, Cambridge, MA, USA, 1988. ISBN 0262010976.

[82] **Sanchez J., Corrales Ramon J. A., BOUZGARROU B. C., and Mezouar Y.** Robotic manipulation and sensing of deformable objects in domestic and industrial applications: A survey. *The International Journal of Robotics Research*, 37:688 – 716, 06 2018.

[83] **Scholkopf B. and Smola A. J.** *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001. ISBN 0262194759.

[84] **Schreiter J., Englert P., Nguyen-Tuong D., and Toussaint M.** Sparse gaussian process regression for compliant, real-time robot control. In *2015 IEEE*

International Conference on Robotics and Automation (ICRA), pages 2586–2591, 2015.

[85] **Siciliano B. and Khatib O.** *Springer Handbook of Robotics*. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 354023957X.

[86] **Siciliano B., Sciavicco L., Villani L., and Oriolo G.** *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 2010. ISBN 1849966346.

[87] **Siciliano B. and Villani L.** *Robot Force Control*. Kluwer Academic Publishers, USA, 1st edition, 2000. ISBN 0792377338.

[88] **Sousa C. D. and Cortesão R.** Physical feasibility of robot base inertial parameter identification: A linear matrix inequality approach. *The International Journal of Robotics Research*, 33(6):931–944, 2014. URL `https://doi.org/10.1177/0278364913514870`.

[89] **Srivastava N., Hinton G., Krizhevsky A., Sutskever I., and Salakhutdinov R.** Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

[90] **Sutton R. S. and Barto A. G.** *Reinforcement learning: An introduction*. MIT press, 2018.

[91] **Takegaki M. and Arimoto S.** A New Feedback Method for Dynamic Control of Manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 103(2): 119–125, 06 1981. ISSN 0022-0434. URL `https://doi.org/10.1115/1.3139651`.

[92] **Terzopoulos D., Platt J., Barr A., and Fleischer K.** Elastically deformable models. *SIGGRAPH Comput. Graph*, 21(4):205–214, 1987.

[93] **Todorov E., Erez T., and Tassa Y.** Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[94] **Urtasun R., Fleet D. J., and Fua P.** 3d people tracking with gaussian process dynamical models. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 238–245. IEEE, 2006.

[95] **Wang J. M., Fleet D. J., and Hertzmann A.** Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence*, 30(2):283–298, 2007.

[96] **Wang J. M., Hertzmann A., and Fleet D. J.** Gaussian process dynamical models. *Advances in neural information processing systems*, 18:1441–1448, 2005.

[97] **Williams C. K. and Rasmussen C. E.** *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006.

[98] **Zhan Q., Liang Y., and Xiao Y.** Color-based segmentation of point clouds. *ISPRS Laser Scanning Workshop*, 38, 07 2009.

[99] **Zhu Z., You X., Yu S., Zou J., and Zhao H.** Dynamic texture modeling and synthesis using multi-kernel gaussian process dynamic model. *Signal Processing*, 124:63–71, 2016.

[100] **Álvarez M., Rosasco L., and Lawrence N.** *Kernels for Vector-Valued Functions: A Review*. 01 2012. ISBN 9781601985590.