SAPIENZA
UNIVERSITÀ DI ROMA

Learning-based methods for Robotic control

Sapienza University of Rome

Dottorato di Ricerca in Automatica, Bioingegneria e Ricerca Operativa – XXXIV Ciclo

Candidate

Giulio Turrisi
ID number 1764899

Thesis Advisor

Prof. Giuseppe Oriolo

January 2022

Learning-based methods for Robotic control
Ph.D. thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: turrisi@diag.uniroma1.it

Abstract

Robots nowadays are being employed in increasingly complex scenarios, where the number of possible assumptions that can be made to ease the control synthesis is getting considerably smaller compared to the past. In fact, back in the day control engineers could heavily rely on a static world assumption and on a perfect knowledge of the system dynamics, since robots were practically only confined in controlled assembly lines where everything was predetermined beforehand. Given these premises, it was fairly easy to synthesize control laws able to solve with high precision the programmed task. Recently, task complexity started to grow considerably with respect to the past, requiring a new type of controller able to adapt continuously to the unknown scenarios to be faced. Among all the new methods, learning-based control can be considered one of the most promising approaches in literature today.

This thesis investigates the use of this new control technique in robotics. We start by giving some background materials on Machine Learning, discussing how we can learn a better dynamical model for the robot just from sensor data, or even directly synthesize a control law from experiences. Then, after a small excursus on Optimal Control we present our contributions in this novel field.

Specifically, a learning-based feedback linearization controller is proposed to deal with model uncertainties in fully actuated robots. This novel technique is then extended to underactuated systems, where control is tremendously complicated by the impossibility in these robots to follow arbitrary trajectories which are not dynamically feasible, i.e. not generated by an exact knowledge of their models.

Finally, we present a contribution in the field of Reinforcement Learning, an approach that is able to learn directly a controller for a given task just by a trial and error mechanism. As detailed in the first chapters, Reinforcement Learning does not assure arbitrary constraints satisfaction in the final learned controller, which limits tremendously its applicability on real platforms. For this aspect, we propose an online mechanism where Optimal Control is used to enhance the safety of the final control law.

Ringraziamenti

Questo percorso è stato certamente non privo di difficoltà, e concluderlo sarebbe stato sicuramente ancora piu difficile senza tutte quelle persone che mi hanno supportato durante questi tre anni. Vorrei quindi ringraziare per tutto il loro aiuto: il mio supervisor di dottorato, Prof. Giuseppe Oriolo, che mi ha guidato in tutti questi anni; i Profs. Leonardo Lanari e Alessandro De Luca, con cui ho avuto il piacere di collaborare in vari lavori; Valerio, Marco C., Claudio, Barbara, Marco F. e Spyros, che sono stati sia ottimi amici che ottimi compagni di lavoro.

Ringrazio infine anche Miriana, che in questi tre anni è stata il mio punto fermo, confortandomi ogni volta che "l'ansia da ricerca" prendeva il sopravvento su di me.

Contents

Chapter 1

Introduction

Historically robots have been always adopted in closed and controlled environments, where the same task was performed thousand of times and the presence of unexpected events was reduced to a minimum. This modality, suitable only for replacing a subset of the repetitive works, greatly simplified the control problem to be faced since the robot model was exactly known in advance, from an offline identification or from the datasheet of the manufacturer, and the environment was predetermined. In fact, usually every object to be manipulated was carefully taken into consideration before the control synthesis, assuring optimal repeatability and accuracy on a given task. Thanks to this simplification and its performance, robots saw a steady increase in usage in the majority of the assembly lines every year (Fig. 1.1), and still nowadays it continues to be the most prominent approach used in industry.

In the last decades, this paradigm started to change due to an increase of demanding tasks to be faced. Robots started to be adopted outside their cage, in what is called a non-structured environment, making it impossible to predetermine exactly a task beforehand. Nowadays autonomous cars are being tested worldwide, robot vacuums are more and more present in houses and new package delivery techniques, such as using drones, are being tested daily in order to optimize costs (Fig. 1.2). All of these examples share an increment in the task complexity with respect to what was requested to the robots in the past, and the same inevitable presence of uncertainties, both in the robot dynamics - which cannot be known exactly in general - and in the environment model. In both cases, the problem

Figure 1.1. Assembly line robots.

Figure 1.2. (left) autonomous driving; (right) drone delivery.

of the control synthesis started to be tremendously complicated, and the number of possible tasks to be solved started to grow up rapidly, requiring specialized controllers. Furthermore, nominal control laws, which were developed under the promise of determinism in the robot model and environment, were shown to be prone to failure in the real world.

Recently, to deal with these new trends in robotics, learning-based controllers have gained popularity in the control community with the promise to deal both with task complexity (*control learning*) and the presence of uncertainties (*model learning*). In the first case, Reinforcement Learning (RL) promises to eliminate the need for hand-tuned controllers favoring generalization [1]. This is achieved by trying to learn an optimal controller to solve the task directly from experience, maximizing a user-tuned reward function and without any - or with a minimum - prior knowledge on the problem. In general, this avoids the need of programming a controller for each new task to be solved, preferring a single and general algorithm that can learn on demand. It can be argued that RL is what humans do in everyday life and that our continuous adaptation is the only way we can deal with the complexity of the world. In robotics, RL has shown great capabilities in different scenarios, for example for solving tasks like grasping [2, 3], navigation [4, 5, 6] and even for legged locomotion [7, 8, 9]. Despite these successful applications, still many challenges remain to be solved that make not straightforward its applicability to real system, such as the need of an excessive number of experiences in continuous and high dimensional state spaces, and the impossibility to analyze precisely its performance or to impose a precise behaviour to the final controller [10]. The latter will be one of the topics discussed in this thesis.

In the second case, uncertainties, that are inherent in a real-world scenario, can be learned in few seconds directly during operation, augmenting continuously a nominal controller with new information about the world. In general, this idea eliminates the need for a static world assumption, and in contrast to the past, it is not needed anymore to know a-priori the time-varying dynamics of a robot, for example caused by wearing, or to know in advance every object dynamical parameters. In the past, two main classes of controllers were used to counteract this unavoidable problem, namely robust [11] and adaptive control [12]. Robust control starts with the assumption of a known and bounded uncertainty set, but although effective, this approach leads to an excessive conservatism in the control law, which in this case should be robust enough to handle all the possible disturbances, and it is corrupted by the presence of high chattering in the control signal. Adaptive control, on the other hand, tries to increase the controller performance via an online estimation of some free parameters in the control law, which are adapted during operation taking into account an error feedback directly from the robot sensors. Therefore, adaptive control can react quickly on the single realization of the uncertainties set increasing performance. Still, this control techniques suffer some drawbacks, such as poor convergence properties to the true uncertainties values [13], hence providing only a local improvement in the performance. On the other hand, learning-based controllers aim to directly estimate the true unknown dynamics before the control synthesis, taking into consideration all the datapoint collected by the sensors during operation. In literature, many outstanding examples of model learning can be found in the context of robotics. For example in [14, 15] Neural Networks are employed to learn the nonlinear dynamics respectively of a drone and an helicopter. In [16] the authors propose an optimal controller coupled with a popular nonparametric regression method, called Gaussian Process, which we will see is considered one of the best tools in the context of online learning, for obtaining superior performance in a car racing scenario. Finally, in [17] multiple local models are concurrently learned for a robotic flexible hand to perform dexterous manipulation using again optimization for robot motions.

Problem statement In this thesis, we tackle the general problem of learning-based control for robotics systems. In the first part, we will present novel algorithms that are able to deal *online* with the problem of uncertainties in model-based control, in which performances are obviously strongly coupled with the exact knowledge of the system. Instead, in the last part of the thesis, we will discuss a method that is able to increase the safety of a learned controller, which is an essential condition for the applicability of RL in a real-world scenario that unfortunately nowadays cannot be assured by any standard learning techniques. Extensive simulations and in many cases experiments will be presented in the following chapters to prove the effectiveness of the proposed methods, which can be applied both for fully-actuated and underactuated robots.

1.1 Manuscript overview

This thesis is organized into seven chapters shortly described below, including this introduction and some final remarks.

- **Chapter 2 - Background on Robot Learning** This chapter provides a review of various learning-based methods for robotics, describing the distinction between the concepts of *model* and *control* learning. Furthermore, some background materials on Machine Learning will be presented in order to make the thesis self-contained. Specifically, two different regression techniques that can be used both for learning a model and for representing a controller are detailed, namely Neural Networks and Gaussian Processes. Furthermore, in the context of RL we will make a digression on two popular solution techniques, called Policy Search and Value function methods.

- **Chapter 3 - Background on Optimal Control** In this chapter, we provide some background material on Optimal Control, which is extensively used in this thesis both for planning and control purposes for various robots. We will explain first what is an optimization problem, presenting a distinction between cases where we cannot impose any constraints on the final control law and cases where this is possible. Specifically, we will present for the first case the *Linear Quadratic Regulator*, a very common controller in underactuated systems, while for the second case we will detail *Model Predictive Control*, both for linear and for nonlinear systems. Finally, for this last controller, we will discuss how it can be efficiently solved by means of numerical tools since a solution cannot be found in general in closed form.

- **Chapter 4 - An online learning procedure for feedback linearization control** The problem of uncertainties in model-based control is characterized in the case of fully-actuated robots. These uncertainties in dynamical parameters, such as mass, inertia, friction, and unknown payloads, can cause the controller to behave unexpectedly, leading to sub-optimal performance. In [18] we proposed an online learning-based version of a feedback-linearization controller that is able to cope with such uncertainties online, improving the robot tracking performance in just a few seconds. The proposed approach was tested in simulation on a Kuka LBR robot and presented during the Conference of Robotics Learning in 2019.

- **Chapter 5 - Online learning for planning and control of underactuated robots** The problem of uncertainties in the dynamics model is studied in the case of underactuated robots, where their effect is seen not only in the low performance of the controller but even during planning. In fact, every planned trajectory needs to be dynamically feasible to be perfectly tracked in those systems, which turns out to be a strong requirement in the presence of even the smallest source of uncertainty. In this case, it is not only needed to optimize the control law in order to increase the control accuracy but it is necessary to plan an *a-priori* accessible trajectory for the robot. In [19] we proposed an iterative approach, composed of sequences of planning and control phases, in order to cope with the aforementioned problem. The proposed approach was tested experimentally on a two-link underactuated robot, the Pendubot, and it was published in the IEEE Robotics and Automation Letters in 2022.

- **Chapter 6 - Enforcing constraints over learned policies via Nonlinear MPC** The problem of constraints satisfaction in learned control policies is studied. In general, these requirements cannot be hardly imposed, neither during the training of the policy nor the deployment on a real robot, due to the unconstrained optimization nature of RL. In [20] we proposed an online safety filter, based on the Real-time Iteration Model Predictive Control, that is able to impose arbitrary constraints on the system while performing the desired task. This work was again experimentally tested on the Pendubot and presented at IFAC in 2020.

- **Chapter 7 – Conclusions** Final remarks on the given contributions are summarized.

Additionally, in the Appendix after the final chapter is detailed the computation of the Pendubot dynamical model, along with the value of its kinematic and dynamic parameters.

Chapter 2

Background on Robot Learning

Robot Learning is a branch of robotics that originates from the intersection of Control Theory and Machine Learning. The main building block is data, recovered by sensors such as cameras, encoders, or inertial measurement units, which measuring the state of the robot and the state of the environment feeds the main feedback loop for all the learning algorithms in this field.

Robot Learning can be divided in different sub-branches, such as *model* and *control learning*. The first, described in Sec. 2.1, is focused on learning directly the model of a system. Most of the time we will start from a nominal knowledge of our robots, which will be then refined during operation comparing its prediction with how the system actually evolves. Model learning makes extensive use of regression techniques from the field of Machine Learning, in order to obtain algorithms with bounded reconstruction error, probabilistic with an associate variance value that defines uncertainties, or in the particular robotics context, that are data-efficient in order to learn on the fly during operation. The second branch, control learning, has the goal of learning directly a controller for a given system and a given task. This is usually done by a trial and error mechanism, having as a feedback not only the robot state from sensors but also a reward function, which embeds the specification of the task to be solved. In this case, we talk about Reinforcement Learning, which will be described in depth in Sec. 2.2. As we will see, RL can be subdivided into two main approaches, namely *value function* methods, where we try to learn a score function that helps us find the optimal control policy, and *policy search* where we directly try to obtain the optimal controller from data. It should be noticed that in literature other algorithms and sub-branches exist in the field of Robotics Learning, such as Imitation Learning, where we try to copy a particular controller, or Inverse Reinforcement Learning, where we try to infer the optimal reward function that in some cases is difficult to synthesize manually, such as in the case of car driving. In both cases, we also have as a feedback the behaviour of an optimal teacher that guides the learning algorithm.

A sketch of the Robotic Learning taxonomy can be found in Fig. 2.1. Many of the terms presented in the figure will be clear after reading the rest of this chapter.

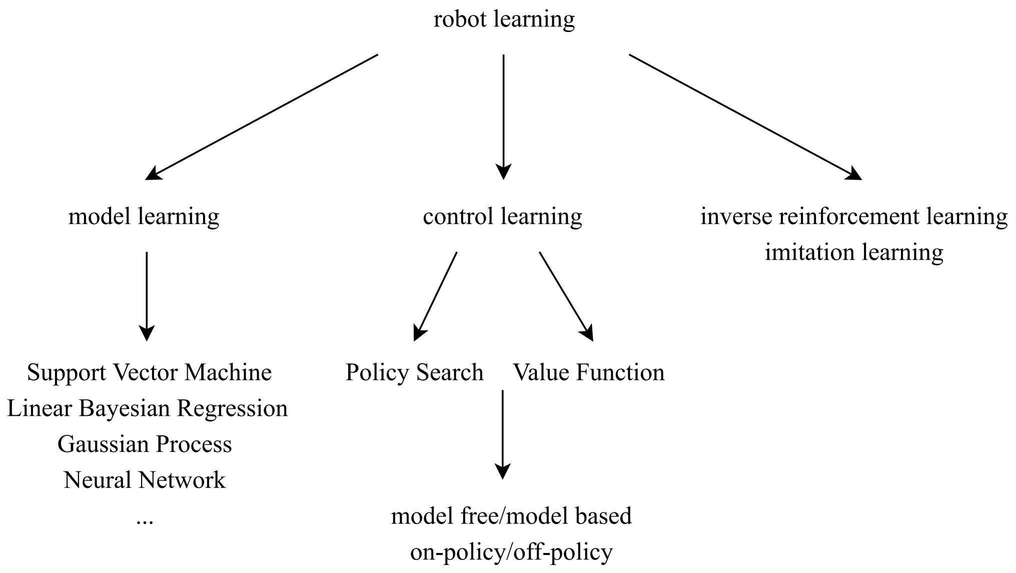

Figure 2.1. Robotic Learning taxonomy. Note that sometimes the division between the branches is not always sharp, e.g. imitation learning can be considered linked to control learning in the final goal but it requires a different dataset.

2.1 About model learning

Models are of paramount importance in robotics. They can be used to predict the behaviour of a system, study its properties analytically or through simulation, or synthesize a particular control law. In general, as we discussed in the introduction, not always a model of a system, found in general by first-order principles, is exact. This can happen when uncertainties are present in the environment, or when the robot is characterized by a time-varying dynamics, for example due to the wearing of its mechanical parts. Other times, a system can even be difficult to model from physics, such as in the case of contact for legged robots, turbulence models, or pedestrian behaviour. For these reasons, nowadays model learning is an attractive method in the robotics community since it gives the possibility of autonomously learn - or correct - a model of the system just by looking at the continuous stream of data coming from the robot sensors.

Many approaches were developed in the last decades showing outstanding performances. In [21], the authors proposed a learning-based robust controller able to decouple robustness from performance, using a Tube-Based Robust Model Predictive Control for the constraints and a learned linear model for the cost function. In [22] a stochastic controller was proposed for a mobile robot, making use of a probabilistic regression model to reduce the conservatism of the control law taking into consideration the uncertainty in the prediction. This is similar to what was done in [23] in the case of a robot manipulator. Finally, in [24] a Neural Network based regression model was employed for developing a nonlinear controller with stability guarantee in order to counteract the aerodynamic effects acting on a drone. In all of these approaches, the first step is to choose the best regression tool from Machine Learning to perform the estimate of the unknown part of the robot dynamics. Different techniques were developed during the years, such as linear model estimation via Least Square (LS), Support Vector Machines (SVM), Gaussian Processes (GP), Neural Networks (NN), and many others. All of them can be chosen *ad-hoc* depending on the characteristics of the unknown function to estimate, which can be linear or nonlinear, the amount of data available for training, and the computational time they require to perform a prediction, which is an important aspect for real-time control application. In general, it is clear looking at the literature that GP and NN, that will be discussed in Sec. 2.1.1 and 2.1.2, are nowadays the prominent methods used in robotics since they are able to estimate *any* nonlinear function without requiring any explicit knowledge of its mathematical form.

Note that in literature there exist two different ways to approach model learning: by learning the direct model or by reconstructing an inverse representation of the robot's dynamic. In the first case (*direct model learning*) the aforementioned method tries to understand how the system, given its actual state, responds to a certain input, while the second approach focus on estimating the input that needs to be given to the system in order to achieve a certain desired new state (*inverse model learning*). Even if one can state that inverse model learning is more similar to control learning in the goal, they are both considered in the literature part of model learning techniques, hence they will be presented here without any major distinction.

2.1.1 Gaussian Process

To understand how we can learn the model of a dynamical system, both in the case of direct or inverse modeling, let us consider first the following generic function

$$y = f(\boldsymbol{x}) + \epsilon \tag{2.1}$$

where $\boldsymbol{x}$ is the input vector with n_x components, y represents the scalar output value which can be measured, for example, by a sensor and ϵ an additive noise with zero mean and variance σ^2. Suppose we have at our disposal a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) \,|i = 1, \dots, n_d\}$ with n_d the number of elements. We wish to infer a function that represents the best our dataset $\mathcal{D}$, called $f_* = \boldsymbol{\theta}^\top \boldsymbol{x}$, where $\boldsymbol{\theta}$ represent a set of parameters that we wish to optimize. Different techniques exist to perform this computation, called regression, that works both in the presence of a linear or nonlinear function f. We will first describe a popular probabilistic method called Bayesian Linear regression (BLr), which will be then augmented with nonlinear input features. BLr will be used to derive lather another popular regression technique in robotics, called Gaussian Process regression (GPr), which will be extensively used in Ch. 4 and 5.

Starting with the assumption of an identical distributed Gaussian noise with a zero mean $\epsilon = \mathcal{N}(0, \sigma^2)$, we are interested, given our dataset $\mathcal{D}$, to compute the best set of parameters $\boldsymbol{\theta}$. The *posterior probability* of the parameters can be written by using Bayes' Theorem as

$$p(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{Y}) = \frac{p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{Y}|\boldsymbol{X})} \tag{2.2}$$

where $p(\boldsymbol{\theta})$ is, by assumption, a Gaussian *prior* over the parameters value $\mathcal{N}(\boldsymbol{m}_p, \boldsymbol{\Sigma}_p)$, $p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta})$ is the *likelihood* of our dataset output value $\boldsymbol{Y}$ given the input data $\boldsymbol{X}$

and the parameters $\boldsymbol{\theta}$, and $p(\boldsymbol{Y}|\boldsymbol{X})$ is the *marginal likelihood*, needed to normalize the posterior. Notice that the dimension of $\boldsymbol{X}$ is $n_d \times n_x$, where n_x represent the dimension of the input vector $\boldsymbol{x}$, while for $\boldsymbol{Y}$ is $n_d \times 1$ since for simplicity we are supposing a scalar output value. The posterior probability distribution of the parameters given the training data can be calculated in closed form obtaining

$$p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{Y}) = \mathcal{N}(\boldsymbol{m}_n, \boldsymbol{\Sigma}_n) \tag{2.3}$$

with $\boldsymbol{m}_n$ and $\boldsymbol{\Sigma}_n$, the mean and variance of the best parameters, equal to

$$\boldsymbol{\Sigma}_n = (\boldsymbol{\Sigma}_p^{-1} + \sigma^{-2}\boldsymbol{X}^\top\boldsymbol{X})^{-1} \tag{2.4}$$

$$\boldsymbol{m}_n = \boldsymbol{\Sigma}_n(\boldsymbol{\Sigma}_p^{-1}\boldsymbol{m}_p + \sigma^{-2}\boldsymbol{X}^\top\boldsymbol{Y}) \tag{2.5}$$

In order to perform a prediction on a given query points $\boldsymbol{x}_*$, we should average the outputs that we obtain for *all* the possible realizations of the parameters $\boldsymbol{\theta}$ that can be extracted by the Gaussian in eq. (2.3). This can be done through the marginalization of $\boldsymbol{\theta}$ over the full probabilistic model $p(y_*|\boldsymbol{x}_*,\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{Y})$, obtaining the posterior predictive distribution

$$p(y_*|\boldsymbol{x}_*,\boldsymbol{X},\boldsymbol{Y}) = \int p(y_*|\boldsymbol{x}_*,\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{Y})d\boldsymbol{\theta} \tag{2.6}$$

$$= \mathcal{N}(\boldsymbol{x}_*^\top\boldsymbol{m}_n, \boldsymbol{x}_*^\top\boldsymbol{\Sigma}_n\boldsymbol{x}_* + \sigma^2). \tag{2.7}$$

See [25] for the complete derivation of eq. (2.7).

In general, when we make a prediction we do not care about the variance, but in some cases, it can be used as an estimate of the regressor uncertainty on a given query point $\boldsymbol{x}_*$. This can enable cautious behaviour in the context of robotics in regions where we do not know enough about our model.

If the function f is nonlinear, the linear model used in the previous computation is prone to fail. A popular workaround to solve this problem is to project the input data $\boldsymbol{x}$ to an higher dimensional space, using a set of nonlinear basis function such as $\boldsymbol{\phi}(\boldsymbol{x}) = (1, \boldsymbol{x}, \boldsymbol{x}^2, \boldsymbol{x}^3, ...)^\top$, which, if we considered from now on a scalar input data x, takes dimension of $1 \times n_\phi$. This set of basis functions can be usually found from physics, as in the case of robotics when we know explicitly the form of the robot dynamics. Repeating the same computation as before (see [25]), we can obtain the equivalent nonlinear BLr model for a scalar input x

$$p(y_*|x_*,\boldsymbol{X},\boldsymbol{Y}) = \mathcal{N}(\boldsymbol{\phi}(x_*)^\top\boldsymbol{m}_n, \boldsymbol{\phi}(x_*)^\top\boldsymbol{\Sigma}_n\boldsymbol{\phi}(x_*) + \sigma^2) \tag{2.8}$$

where, inside $\boldsymbol{m}_n$ and $\boldsymbol{\Sigma}_n$ the input vector data $\boldsymbol{X}$ is substituted with $\boldsymbol{\Phi}$, representing the corresponding matrix obtained computing the basis function for all the $\boldsymbol{X}$ elements. Hence, the mean and the variance are defined as below

$$\boldsymbol{\Sigma}_n = (\boldsymbol{\Sigma}_p^{-1} + \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1} \tag{2.9}$$

$$\boldsymbol{m}_n = \boldsymbol{\Sigma}_n(\boldsymbol{\Sigma}_p^{-1}\boldsymbol{m}_p + \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{Y}). \tag{2.10}$$

GPr can be seen as a generalization of BLr with an infinite number of basis function. We can rewrite eq. (2.8), substituting inside the elements of eqs. (2.9-2.10) as

$$\begin{gathered}\mathcal{N}(\boldsymbol{\phi}_*^\top\boldsymbol{\Sigma}_p\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Sigma}_p\boldsymbol{\Phi} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{y},\\ \boldsymbol{\phi}_*^\top\boldsymbol{\Sigma}_p\boldsymbol{\phi}_* - \boldsymbol{\phi}_*^\top\boldsymbol{\Sigma}_p\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Sigma}_p\boldsymbol{\Phi} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Sigma}_p\boldsymbol{\phi}_*)\end{gathered}$$

highlighting all the multiplication which involves the basis functions $\boldsymbol{\phi}_*$ and the matrix $\boldsymbol{\Phi}$. Each of them can be split into multiple dot products and represented by a function $k(x_i, x_j)$, usually called covariance function or kernel, where the indexes i and j represent a generic query or data point in the dataset $\mathcal{D}$. Replacing the calculation of the basis functions with a single kernel can be convenient since it can free from the construction of hand-tuned features favouring generalization. For example, a popular choice for $k(\cdot,\cdot)$ is the exponential (or gaussian) kernel, defined as

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2}\right) \tag{2.11}$$

which is shown to be equivalent to an *infinite* dimensional set of basis functions. Usually, this kernel is used in conjunction with some hyperparameters to be optimized, e.g. in the case of eq. (2.11) we have

$$k(x_i, x_j) = a^2 \exp\left(-\frac{\|x_i - x_j\|^2}{2\, l^2}\right) \tag{2.12}$$

where l and a are called the length-scale and the amplitude of the kernel. Other kernel functions used in literature are for example

- linear: $k(x_i, x_j) = x_i x_j$
- linear with features: $k(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$
- rational quadratic: $k(x_i, x_j) = a^2 \left(1 + \frac{\|x_i - x_j\|^2}{2\alpha\ell^2}\right)^{-\alpha}$, where α is a scaling factor
- periodic: $k(x_i, x_j) = \exp\left(-2\sin^2\left(\frac{x_i - x_j'}{2}\right)\frac{1}{l^2}\right)$

and each of them can be chosen depending on our prior knowledge of the function f to estimate, e.g. if we know that it is periodic we can opt for the last kernel on the list, instead if we want to incorporate some ad hoc features we can choose the second one. Kernel function can be even hand-tuned, but they should respect some properties such as

- symmetry: $k(x_i, x_j) = k(x_j, x_i)$
- positive-definiteness: $\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k\left(x_i, x_j\right) \geq 0 \quad \forall n \in \mathbb{N}$ and $\forall c_i, c_j \in \mathbb{R}$

Furthermore, given that the sum or the product of two kernels is still a valid kernel, one can usually mix them at need.

For prediction, we can compute the mean and the variance of a GP on a query point x_* as

$$f_* = \boldsymbol{k}(x_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I})^{-1}\boldsymbol{Y} \tag{2.13}$$

$$\sigma_* = k(x_*, x_*) - \boldsymbol{k}(x_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I})^{-1}\boldsymbol{K}(\boldsymbol{X}, x_*) \tag{2.14}$$

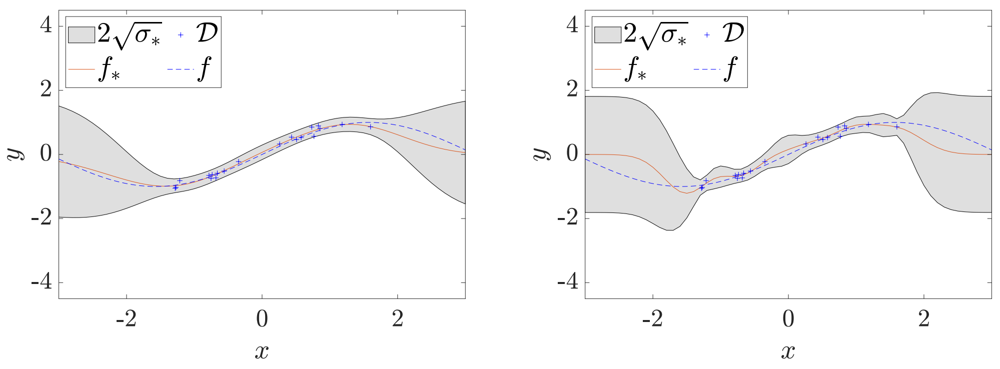

Figure 2.2. Example of GP regression varying the hyperparameters of an exponential kernel with an optimized lenght-scale $l = 0.9048$ (left), and with a non-optimized value of $l = 0.3679$ (right). In both cases, the dashed blue line represent the true function $f(x) = sin(x)$, while in red is plotted the mean prediction f_* (eq. 2.13) and in grey the 95% confidence interval obtained using the variance information of eq. (2.14) plus and minus the mean. Furthermore, in blue and using the plus symbol are plotted the training points corrupted by a small additive noise.

where $\boldsymbol{k}(x_*, \boldsymbol{X})$ is a vector of dimension $1 \times n_d$ where each component is the output of the kernel function (2.11) between x_* and the entire dataset $\boldsymbol{X}$, and $\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X})$ is the equivalent kernel matrix with dimension $n_d \times n_d$.

Finally, is important to mention that the set of hyperparameters in (2.11) are in general tuned through gradient descent, maximizing the marginal likelihood equation over *all the possible* functions f represented by the GP

$$p(\boldsymbol{Y}|\boldsymbol{X}) = \int p(\boldsymbol{Y}|f, \boldsymbol{X}) p(f|\boldsymbol{X}) df \tag{2.15}$$

where $p(f|\boldsymbol{X}) = \mathcal{N}(0, \boldsymbol{K})$ is the GP prior (for simplicity with mean 0) and $p(\boldsymbol{Y}|f, \boldsymbol{X}) = \mathcal{N}(f, \sigma_n \boldsymbol{I})$ is the likelihood. To facilitate the optimization, eq. (2.15) is transformed conveniently in the log form

$$-\frac{1}{2}\boldsymbol{Y}^T \left(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2 \boldsymbol{I}\right)^{-1} \boldsymbol{Y} - \frac{1}{2}\log\left(\left|\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2 \boldsymbol{I}\right|\right) - \frac{n_d}{2}\log(2\pi)$$

which derivation can be found in [26], Ch. 2. The three terms in eq. (2.1.1) are dependant respectively on the training data (first term), on the complexity of the model defined on the covariance matrix $\boldsymbol{K}$, and on a normalizing constant (the last term). In Fig. 2.2 is reported the regression result obtained from two GPs, with and without optimizing their hyperparameters. As can be seen from the figure, decreasing the length-scale l of the kernel (2.12) we can augment the flexibility of the learned regressor which become more rough.

Additionally to the maximization of eq. (2.1.1), we can perform a *cross-validation* step on our dataset $\mathcal{D}$. The basic idea is to split the dataset into two disjoint sets, one for training while the other to be used as a validation set, to monitor performance such as the generalization error on the chosen kernel function. In practice, a drawback of the hold-out method is that only a fraction of the full dataset can be used for training and that if the validation set is small, the performance estimate obtained may have a large variance. Another drawback is that just performing an optimization step is usually computationally expensive on its own since it requires an inversion of the covariance matrix $\boldsymbol{K}$, and cross-validation adds another costly computation. For these reasons, usually in online model learning this is not carried out since we seek to maximize speed. Another potential issue for real-time control is that the computational complexity of the prediction of a GP is $\mathcal{O}(n_d^3)$, with n_d the size of the dataset. To keep the computation of the prediction fast enough, different approximations exist which use only a reduced set of only d datapoints, chosen for example on the basis of the information gain criterion [27], or more commonly in the case of robotics chosen depending on the distance from the reference trajectory.

The readers can refer to [26] for a broader treatment about the GP theory and their training procedure, and for a more detailed list of the different types of kernel functions that can be adopted.

A noticeable example of a learning-based controller using GP can be found in [28], where different techniques are detailed in order to propagate the regressor inside a Nonlinear Model Predictive Control, together with a description on how probabilistic bounds for the controller constraints can be generated according to the variance information (eq. 2.14). The GP mean and variance are again used in [29], where the authors proposed a procedure to tune the weights of a PD controller to counteract model uncertainties, and in [30] where a Sliding Mode Controller is adopted using the variance directly in the discontinuous part of the control law. Many other works in literature have shown that GPs are a viable solution in the case of unknown dynamics and that they can be used for a large variety of different controllers, for example in Differential Dynamic Programming [31], Adaptive Control [32] and finally Backstepping [33].

2.1.2 Neural Network

Neural Network (NN) is another popular technique in the Machine Learning community used for solving regression problems. So far we have considered model of the form $f(\boldsymbol{x}) = \phi(\boldsymbol{x})^\top \boldsymbol{\theta}$, where $\phi(\boldsymbol{x})$ can be some combination of nonlinear features in order to perform nonlinear regression. In the case of NN, we can perform the same task starting from a concatenation of a simple equation, representing a *neuron* and expressed as

$$\phi(\boldsymbol{x}) = \sigma(\boldsymbol{x}^\top \boldsymbol{W} + \boldsymbol{b}) \tag{2.16}$$

where σ are some nonlinear functions that perform computation component-wise and output a single scalar value, and $\boldsymbol{W}$ and $\boldsymbol{b}$, called weights and bias, are the parameters $\boldsymbol{\theta} = [\boldsymbol{W}, \boldsymbol{b}]$ to be tuned. Commonly used nonlinear functions σ are for example

- sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- relu: $\sigma(x) = \max(0, x)$
- tanh: $\sigma(x) = \frac{e^{2x}+1}{e^{-2x}-1}$

but many others can be found in literature.

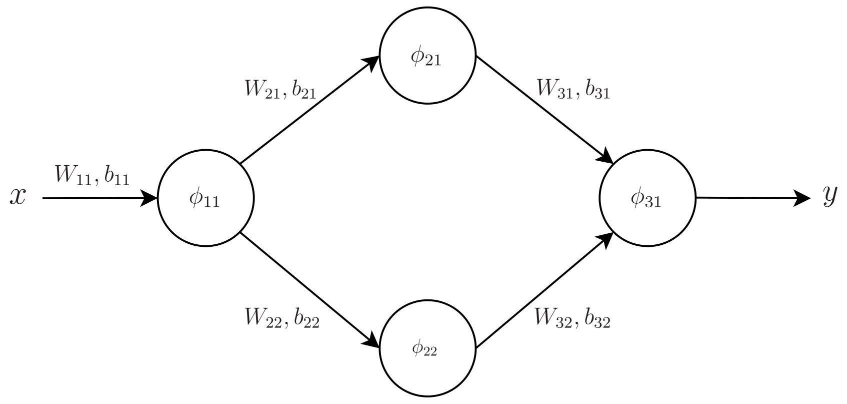

Figure 2.3. A simple three-layers Neural Network, with a single neuron both in the input and in the output layer.

A NN is usually composed by a concatenation of multiple neurons, each of them represented by the eq. (2.16). For example, the output of a single neuron can enter as input to another one, forming a *two-layers* NN

$$\phi_{21}(\boldsymbol{x}) = \sigma(\phi_{11}(\boldsymbol{x})^\top W_2 + b_2)$$

where the first number in the subscript identify the layer while the second the neuron. Furthermore, multiple neurons can coexist in parallel in a single layer, e.g.

$$\phi_{31}(\boldsymbol{x}) = \sigma(\phi_{21}(\boldsymbol{x})^\top W_{31} + b_{31} + \phi_{22}(\boldsymbol{x})^\top W_{32} + b_{32})$$

where both $\phi_{21}(\cdot)$ and $\phi_{22}(\cdot)$ have the same input, e.g. the original data or another neuron itself, and $\boldsymbol{W}_3 = [W_{31}, W_{32}]$, $\boldsymbol{b}_3 = [b_{31}, b_{32}]$ are partitioned accordingly.

The most common architecture for a NN is composed of three layers, represented in Fig. 2.3, where the first is called the *input layer* and takes as input the data $\boldsymbol{x}$, the second is called the *hidden layer*, while the third is the *output layer* that finally outputs the value of the reconstructed function f. It is important to know that the use of a three-layer NN is due to the universal approximation theorem [34], which states that an architecture composed by a linear output layer and at least one hidden layer with a nonlinear activation function, is able to approximate *any* function, provided that the hidden layer is composed by *enough* parallel neurons. We are not guaranteed, however, that the training algorithm, which will tune the weights and bias $\boldsymbol{W}$ and $\boldsymbol{b}$, will be able to learn that function. This can fail for a variety of reasons, such as the optimization algorithm used for training may not be able to find the optimal parameters that correspond to the true function, or we may have not chosen the right amount of hidden units. In fact, the theorem does not state the exact amount of neurons needed, and one in practice usually prefers to increase the number of hidden layers until the needed regression accuracy is reached.

To describe the training procedure of a NN, we can start first defining a loss function over the entire dataset $\mathcal{D}$ defined previously in Sec. 2.1.1. For example, for a least-square loss we have

$$L(\boldsymbol{\theta}) = \frac{1}{n_d} \sum_{i=0}^{n_d} (y_i - f_{NN}(x_i)) \tag{2.17}$$

where f_{NN} is the output of the NN queried at the known datapoint x_i. From eq. 2.17 we can compute the gradient over the parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ and perform a gradient descent step. For example, for the weights and the bias at the layer j we have the following update rule

$$\boldsymbol{W}_j = \boldsymbol{W}_j - \alpha \frac{\partial L}{\partial \boldsymbol{W}_j} \tag{2.18}$$

$$\boldsymbol{b}_j = \boldsymbol{b}_j - \alpha \frac{\partial L}{\partial \boldsymbol{b}_j} \tag{2.19}$$

where α is a chosen learning rate used to stabilize the training procedure.

For a deeper treatment on Neural Networks, their training procedure, and a description of different architectures such as Convolutional and Recurrent NN, the reader can refer to [35].

In the field of Robot Learning, Neural Networks are used both in the case of model and control learning. In the first case, they show great reconstruction performances and outstanding generalization capabilities, meaning that if the training is performed correctly and the data is informative enough they can converge to the true function f. Furthermore, their prediction time remains constant with the increase of the dataset, differently from GP. An example of their use in robotics can be found in [36], where they were employed to generate a dynamically feasible trajectory for a drone starting from a coarse initial guess, and in [37] where they were used to represent the model of an elastic surgical robot, difficult to represent and to identify using standard techniques. Finally, other works with NN can be found in [38], where they were coupled with Control Barrier Function to attain a safe control law, and in [39] where a new interesting NN architecture, based on physics principles, was proposed for model learning.

Despite their capabilities, NNs are not considered on par with Gaussian Processes in the field of online learning, i.e. when we are in a low data regime and we have a very little amount of data given by the robot. In fact, NNs are considered more *data-hungry*, and furthermore they do not automatically embed any variance information (eq. 2.14), that can be used for example to obtain a safer control law. Even so, they have a broader usage in the context of control learning since they can be used to represent practically an infinite number of possible nonlinear controllers.

2.2 About control learning

Control learning is at the forefront of research in Robot Learning nowadays, and his biggest representative, Reinforcement Learning [40], in recent years has been proven to be particularly successful in extremely complex application scenarios, such as from beating professional gamers [41, 42], dexterous in-hand manipulation for solving a Rubik's cube [43] to locomotion problems both for quadruped [44] and humanoid [45] robots.

The goal of RL is to directly learn an optimal controller, defined as $\boldsymbol{\pi}$, through experience and interactions with the environment. During each training episode, at each control step k the robot sense its state $\boldsymbol{x}_k$, choose a control action $\boldsymbol{u}_k = \boldsymbol{\pi}(\boldsymbol{x}_k)$ and receive as feedback a reward signal $r(\boldsymbol{x}_k, \boldsymbol{u}_k)$ that represent how optimal is its choice. Shaping the reward r is not always straightforward, since it should embed both the desired task to be solved and the behaviour to be adopted. For example, in the case of a robotic manipulator in a reaching task, r can be composed by a term that represents the distance of its end-effector from the object to grasp, minus some terms that discourage collisions and minimize torques. Along with the reward signal, the robot receives also its next state $\boldsymbol{x}_{k+1}$, making the knowledge of its model not always necessary.

RL can be seen as an unconstrained optimization problem in which we seek to maximize - not minimize for an historical reason - the future rewards, described as

$$J(\boldsymbol{x}_k, \boldsymbol{u}_k) = \sum_{k=1}^{\infty} r(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{2.20}$$

where, in general, this summation is truncated after N steps in order to represent the maximum length of a given training episode. The necessity of this finite horizon formulation arises from a practical prospective, making the optimization problem more tractable and finite dimensional. Usually, the robot can be trained both in simulation or in the real world, and after the end of the horizon is brought back to its initial state. This helps the to encounter *multiple* similar experiences from which we can extract meaningful informations about the task.

RL is strongly related to the concepts of dynamic programming (DP), which uses as a backbone the *Bellman's principle of optimality* [46], which alleges that each subtrajectory of an optimal trajectory is an optimal trajectory as well. In DP we try to find the optimal policy minimizing backward in time the value function

$$J_k(\boldsymbol{x}_k) = r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \min_{\boldsymbol{u}_k} J_{k+1}(f(\boldsymbol{x}_k, \boldsymbol{u}_k)) \tag{2.21}$$

starting from a known final cost $J_N(\boldsymbol{x}_N)$ at end of the horizon. Eq. (2.21), in the case of linear systems, can be optimized in closed form in order to derive an optimal controller such as the discrete Linear Quadratic Regulator (LQR), or for the nonlinear case controllers such as Differential Dynamic Programming [47] and its lightweight variant Iterative LQR [48], both of which have proven to be very successful in robotics in a variety of different complex scenarios [49]. In RL in general we do not know the form of the value function J_i, nor the model of the system f. Hence we cannot make use directly of DP in order to optimize directly our control policy.

2.2.1 Value function approximation

A first workaround for this problem comes from the *value function methods*, in which we try to estimate the value function J_i directly from data. More specifically, we are usually interested in a surrogate function $Q(\boldsymbol{x}, \boldsymbol{u})$, called state-action value function or Q-value function, defined as

$$Q^{\boldsymbol{\pi}}(\boldsymbol{x}_k, \boldsymbol{u}_k) = r(\boldsymbol{x}_k, \boldsymbol{u}_k) + J_f^{\boldsymbol{\pi}}(\boldsymbol{x}_{k+1}) \tag{2.22}$$

which directly associate a *score* to the choice of a particular control action $\boldsymbol{u}_k$ in a given state $\boldsymbol{x}_k$, and then following the actual policy $\boldsymbol{\pi}$. If we can estimate correctly the Q-value function, and we have in hand a finite dimensional problem with a finite number of control actions, just taking the best action with the greatest associate Q value can give us back the optimal control policy. In order to estimate it, different algorithms were developed during the years, such as SARSA [50] and Q-Learning [51], both of which will be briefly described here for clarity.

Let us consider first a discrete scenario, with a finite number of possible states $\boldsymbol{x}_i$, with $i = 0, ..., I$ and a finite number of control actions $\boldsymbol{u}_j$, with $j = 0, ..., J$, e.g. a grid-world where we can just move in four different directions such as up, down, left and right. In SARSA (State-Action-Reward-State-Action), we can define the Q-value function in a tabular form, with dimension $I \times J$. In order to optimize it, we start from a *zero* initial guess in all its cells, and then we incrementally perform an update following the equation

$$Q^{\pi}(\boldsymbol{x}_k, \boldsymbol{u}_k) = Q^{\pi}(\boldsymbol{x}_k, \boldsymbol{u}_k) + \alpha(r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \lambda Q^{\pi}(\boldsymbol{x}_{k+1}, \boldsymbol{u}_{k+1}) - Q^{\pi}(\boldsymbol{x}_k, \boldsymbol{u}_k)) \quad (2.23)$$

where α is called learning rate, and define how much we modify our current guess of Q, and λ is a discount factor $\in [0, 1)$ which tunes the importance of future rewards. With a $\lambda = 0$, the agent will be myopic by only considering the instantaneous reward r, instead with a value greater than 1 the Q-value function could actually diverge. We use here the subscript k to define a general step in the environment, without worrying about the subscript i and j for simplicity. At each time step the robot senses its state $\boldsymbol{x}_k$ and chooses a control action $\boldsymbol{u}_k$ following an ϵ-greedy policy, which for example alternates between a control action that maximizes Q and a random action for exploration. Analyzing eq. (2.23), we can see that SARSA requires not only the state and the control input at state k but even at the next step $k + 1$, alongside the immediate reward r. Furthermore, it is considered an *on-policy* algorithm, which means that the update of the Q-value function is strongly related to the policy adopted (ϵ-greedy). A sketch of the algorithm SARSA, which can help to understand its generic iteration, can be found in Table 1. The algorithm runs for a finite number of episodes, each of them for a given horizon/number of steps.

Algorithm 1 SARSA algorithm

```
Require: $\alpha > 0$, $\lambda \in [0, 1)$, exploration rate $\epsilon > 0$
1: Initialize $Q^{\pi}(\boldsymbol{x}_i, \boldsymbol{u}_j) = 0$ for all $\boldsymbol{x}_i$ and $\boldsymbol{u}_j$ with $i = 0, ..., I$ and $j = 0, ..., J$
2: Loop for each episode:
3:   initialize the state $\boldsymbol{x}_0$ randomly and set $k = 0$
4:   Loop for each step of the horizon:
5:     choose action $\boldsymbol{u}_k$ using policy derived by $Q^{\pi}$ (e.g. $\epsilon$-greedy)
6:     observe $r(\boldsymbol{x}_k, \boldsymbol{u}_k)$ and $\boldsymbol{x}_{k+1}$
7:     choose action $\boldsymbol{u}_{k+1}$ from $\boldsymbol{x}_{k+1}$ using policy derived by $Q^{\pi}$ (e.g. $\epsilon$-greedy)
8:     $Q^{\pi}(\boldsymbol{x}_k, \boldsymbol{u}_k) = Q^{\pi}(\boldsymbol{x}_k, \boldsymbol{u}_k) + \alpha(r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \lambda Q^{\pi}(\boldsymbol{x}_{k+1}, \boldsymbol{u}_{k+1}) - Q^{\pi}(\boldsymbol{x}_k, \boldsymbol{u}_k))$
9:     set $k = k + 1$
```

Q-learning is an *off-policy* variant of SARSA, meaning that the Q function estimated is not directly hinged to a specific policy followed. In fact, it seeks directly to find the optimal Q-value function Q^* through the following update law

$$Q^*(\boldsymbol{x}_k, \boldsymbol{u}_k) = Q^*(\boldsymbol{x}_k, \boldsymbol{u}_k) + \alpha(r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \lambda \max_{\boldsymbol{u}} Q^*(\boldsymbol{x}_{k+1}, \boldsymbol{u}) - Q^*(\boldsymbol{x}_k, \boldsymbol{u}_k)) \quad (2.24)$$

where, compared to eq. (2.23), we choose at each time step the current guessed *best* action $\boldsymbol{u}_{k+1}$ to maximize the future rewards. The algorithm follows the generic iteration described in Table 2, using eq. (2.24) instead of eq. (2.23) and without the requirement of using of an ϵ-greedy policy to choose the next action $\boldsymbol{u}_{k+1}$. Both SARSA and Q-Learning are proved to converge to the optimal Q-value function Q^* with a rate that depends on the exploration policy used and on α. Both of them do not require a model of the system f, and for this reason, are considered *model-free*.

Algorithm 2 Q-learning algorithm

```
Require: $\alpha > 0$, $\lambda \in [0, 1)$, exploration rate $\epsilon > 0$
1: Initialize $Q^*(\boldsymbol{x}, \boldsymbol{u}) = 0$ for all $\boldsymbol{x}_i$ and $\boldsymbol{u}_j$ with $i = 0, ..., I$ and $j = 0, ..., J$
2: Loop for each episode:
3:   initialize the state $\boldsymbol{x}_0$ randomly and set $k = 0$
4:   Loop for each step of the horizon:
5:     choose action $\boldsymbol{u}_k$ using policy derived by $Q^*$ (e.g. $\epsilon$-greedy)
6:     observe $r(\boldsymbol{x}_k, \boldsymbol{u}_k)$ and $\boldsymbol{x}_{k+1}$
7:     $Q^*(\boldsymbol{x}_k, \boldsymbol{u}_k) = Q^*(\boldsymbol{x}_k, \boldsymbol{u}_k) + \alpha(r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \lambda \max_{\boldsymbol{u}} Q^*(\boldsymbol{x}_{k+1}, \boldsymbol{u}) - Q^*(\boldsymbol{x}_k, \boldsymbol{u}_k))$
8:     set $k = k + 1$
```

Having a model, in general, can speed up significantly the convergence of these algorithms, and in literature many RL algorithms exist which try to learn concurrently both f and Q. These methods are called *model-based*, and are proven to be more *data-efficient* with respect to model-free algorithms, meaning they require far less interaction with the environment to converge. One of the first model-based RL algorithms is Dyna-Q [52], which can be considered a model-based extension of Q-Learning. It not only estimates the optimal Q-value function as done in eq. (2.24) via experience, but concurrently tries to learn a model of the environment/robots which can be used to perform multiple updates of Q entirely offline speeding up the convergence of the algorithm. A summary of the algorithm can be found in Table 3. One can observe that a limitation of this approach (and by extension to all the model-based RL algorithms) is that its performance strongly depends on the accuracy of the learned model f. For this reason, usually in literature model-free algorithms outperform their model-based counterpart.

As we saw, in the case of discrete and low dimensional systems the optimal Q or V can be easily found in tabular form, but in general in robotics we are more interested in high dimensional and nonlinear systems which require the use of some function approximators. For example, using the tool described in Sec. 2.1.1, we can represent the Q-value function using a linear model with nonlinear input features, e.g. $Q = \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u})^\top \boldsymbol{\theta}$, where $\boldsymbol{\theta}$ are the usual parameters to tune. In this case, we can seek for the optimal Q^* iteratively, similar to what was done before, defining the loss function

Algorithm 3 Dyna-Q algorithm

```
Require: $\alpha > 0$, $\lambda \in [0, 1)$, exploration rate $\epsilon > 0$
 1: Initialize $Q^*(\boldsymbol{x}, \boldsymbol{u}) = 0$ for all $\boldsymbol{x}_i$ and $\boldsymbol{u}_j$ with $i = 0, ..., I$ and $j = 0, ..., J$
 2: Loop for each episode:
 3:   initialize the state $\boldsymbol{x}_0$ randomly and set $k = 0$
 4:   Loop for each step of the horizon:
 5:     choose action $\boldsymbol{u}_k$ using policy derived by $Q^*$ (e.g. $\epsilon$-greedy)
 6:     observe $r(\boldsymbol{x}_k, \boldsymbol{u}_k)$ and $\boldsymbol{x}_{k+1}$
 7:     $Q^*(\boldsymbol{x}_k, \boldsymbol{u}_k) = Q^*(\boldsymbol{x}_k, \boldsymbol{u}_k) + \alpha(r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \lambda \max_{\boldsymbol{u}} Q^*(\boldsymbol{x}_{k+1}, \boldsymbol{u}) - Q^*(\boldsymbol{x}_k, \boldsymbol{u}_k))$
 8:     set $k = k + 1$ and append to $\mathcal{D}$ the training data $(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{x}_{k+1}, r)$
 9:     learn a model of the system and of the reward function $\boldsymbol{f}_*(\boldsymbol{x}_k, \boldsymbol{u}_k)$ using $\mathcal{D}$
10:     Loop for each simulated step of the horizon:
11:       choose a random state $\boldsymbol{x}_k$
12:       choose action $\boldsymbol{u}_k$ using policy derived by $Q^*$ (e.g. $\epsilon$-greedy)
13:       query the learned model $\boldsymbol{x}_{k+1}, r = \boldsymbol{f}_*(\boldsymbol{x}_k, \boldsymbol{u}_k)$
14:       update again $Q^*$ using eq. (2.24)
```

$$L(\boldsymbol{\theta}) = \frac{1}{2}(r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \lambda \max_{\boldsymbol{u}} \boldsymbol{\phi}(\boldsymbol{x}_{k+1}, \boldsymbol{u})^\top \boldsymbol{\theta} - \boldsymbol{\phi}(\boldsymbol{x}_k, \boldsymbol{u}_k)^\top \boldsymbol{\theta}) \tag{2.25}$$

and updating the parameters $\boldsymbol{\theta}$ via for example a least-square solution. Unfortunately, the use of function approximation annihilates the proof of convergence of the algorithms described previously, but it is an essential step to deal with more complex RL problems.

Even if linear function or Gaussian Process can be used to learn Q efficiently, nowadays the most used function approximators are Neural Networks. In fact, one of the greatest breakthroughs in RL is dated back to 2013 [53], where Q-Learning and NN were coupled together attaining superhuman performances in a variety of atari games.

2.2.2 Policy Search

The use of value function methods is, in general, troublesome if the action space is continuous, such as in Robotics. In fact, in a discrete setting once we estimate the optimal $Q^*(\boldsymbol{x}, \boldsymbol{u})$ we can choose the best action $\boldsymbol{u}$ just by evaluating all the possibilities, but this cannot be easily done in a continuous action space. Additionally, value function methods in general need global convergence of the value function over the entire state-input space, which can be in general troublesome in a low data regime. Another popular RL method, called *policy search*, tries to bypass these problems by finding directly an optimal policy $\boldsymbol{\pi}$ without estimating directly the optimal Q-value function. In this case, we can parametrize our policy using some function approximators as before, e.g. $\boldsymbol{\pi}(\boldsymbol{x}_k) = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\theta}$, and try to perform a gradient descent step directly over the cumulative reward of eq. (2.20). This can be done starting from a fixed initial state $\boldsymbol{x}_0$ and a fixed set of parameters $\boldsymbol{\theta}_i$, performing multiple experiments with small perturbations of $\boldsymbol{\theta}_i$ and registering the different obtained cumulative rewards J. Then, we can perform a batch update of the form

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \alpha \nabla_{\boldsymbol{\theta}} J \tag{2.26}$$

with

$$\nabla_{\boldsymbol{\theta}} J = (\Delta\boldsymbol{\theta}^\top \Delta\boldsymbol{\theta})^{-1} \Delta\boldsymbol{\theta}^\top \Delta J \tag{2.27}$$

where the matrices $\Delta\boldsymbol{\theta}$ and ΔJ represent the stacked samples of perturbation on the parameters and their corresponding cumulative reward. This update law in general behaves poorly, because it strongly depends on the learning rate α and can be very noisy. Furthermore, it requires a reinitialization of the system in the same initial position $\boldsymbol{x}_0$ in order to make the cumulative reward J comparable between experiences.

A better approximation of $\nabla_{\boldsymbol{\theta}} J$ comes from the *policy gradient theorem* which first application can be found in the algorithm REINFORCE [54]. For a *deterministic* policy it reduces to the deterministic policy gradient theorem [55], where the gradient is simply defined as

$$\nabla_{\boldsymbol{\theta}} J = \nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{x}) \nabla_{\boldsymbol{u}} Q^{\pi}(\boldsymbol{x}, \boldsymbol{u})|_{\boldsymbol{u}=\boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{x})} \tag{2.28}$$

which, roughly speaking, links the direction of improvement of the policy $\boldsymbol{\pi}$ with the direction of increment of the Q-value function. Algorithms that use eq. (2.28) together with function approximators are called *actor-critic*, where the term actor refers to the policy while critic to the estimator of Q.

Even in the case of policy search, algorithms can be subdivided into on-policy and off-policy, which can be both model-free [56] or model-based [57]. The readers can refer to [58] for a broader description of policy search methods.

Applications of PS can be found in all areas of robotics. For example, in [59], a stochastic policy is optimized for solving the locomotion problem for a simple 3D humanoid. In [60], a model-based PS approach is applied to a low cost manipulator showing astonishing data-efficiency, and finally in [61] PS is used as an high level governor for a Model Predictive Controller for maximizing performance on a drone.

2.3 Chapter Summary

This chapter gave an overview of the necessary theoretical foundations for the field of Robot Learning. It began by stating its two main branches, namely *model* and *control learning*, and then providing a description of their core components. In the case of model learning, different regression techniques were detailed, such as Linear weighted Bayesian regression, Gaussian Processes, and Neural Networks. In the case of control learning, RL was introduced by describing two different methodologies (value function method and policy search) to obtain a control policy.

In the next two chapters (Chs. 4–5), Gaussian Processes will be used to describe two novel learning-based algorithms, one that is applicable to fully-actuated robots while the other tailored for underactuated platforms.

Instead in Ch. 6 of this thesis, we will make use of a popular actor-critic, model-free, and off-policy RL algorithm [62]. It should be clear from this chapter that RL is solved by an unconstrained optimization procedure. In the case of discrete and finite state space, we usually obtain the optimal solution, hence if we embed in the reward function some hard constraints as penalties they will be not violated at convergence. This is not true in a continuous state space, such as in robotics, since the optimal solution is not usually found and a sub-optimal solution can lead the robot to act unsafely. This motivates Ch. 6, where an online *safety filter* is presented to mitigate this issue.

Chapter 3

Background on Optimal Control

An appealing method to command motions to a robot is by deriving control laws that are optimal with respect to some chosen criteria, e.g. that minimize for example energy consumption while enabling the system to reach some desired goal. This idea makes the foundation of the field of Optimal Control Theory, which as we will see, shares strong similarities with Reinforcement Learning discussed previously in Sec. 2.2. In fact, RL and Optimal Control can be seen as highly related topics because they actually share the same goal, i.e. obtaining the most optimal controller given, in the first case, a reward function and in the second one a cost. Still, they approach this problem in a very different way: for example, RL does not explicitly require to know neither the final form of the controller to be optimized nor the model of the system to control, while in contrast, in Optimal Control is essential to define both of them beforehand.

Nowadays Optimal Control is a well established topic in robotics for obtaining powerful linear and nonlinear control laws, and the aim of this chapter is to present a series of techniques that will be used in the rest of the thesis. In Sec. 3.1, we will derive a linear optimal controller called Linear Quadratic Regulator (LQR), used in Chs. 5–6 to stabilize the Pendubot around its equilibrium points. In Sec. 3.2 a popular control technique able to deal with state and inputs constraints, called Model Predictive Control (MPC), is introduced both for linear and nonlinear systems. MPC and Nonlinear MPC (NMPC) will be used respectively in Chs. 4–5. Unfortunately, NMPC cannot be used as it is for control since it introduces a large delay due to its expensive computation. For this reason, in Sec. 3.3 we will introduce the Real Time Iteration (RTI) scheme, a popular control method that is able to obtain real-time nonlinear control laws. RTI will be used in Ch. 6 to generate an online safety filter for learned control policies. Finally, since most of these controllers cannot be solved in closed form, in Sec. 3.4 we will describe a popular numerical solver.

3.1 Linear Quadratic Regulator

LQR is a linear controller widely used even in the context of nonlinear systems. In fact, usually one can perform a linearization of the dynamics equation of the system near an equilibrium point, obtaining as a result an equivalent linear representation that is valid only in the vicinity of the chosen point. This procedure is very common since from Control Theory we have well established tools to synthesize linear control laws to stabilize a system. The synthesis of LQR starts from the choice of the cost function to minimize. This is usually chosen to be in a quadratic form since, if the model of the system is linear, we will obtain a Quadratic Problem (QP) that is easier to solve both in closed form or using numerical methods. In fact, in a QP we have only one local solution that is also the global optimum of the problem, hence we do not have to perform an expansive search of the best solution in the optimization problem. For simplicity, we start from a linear and *discrete* model of the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k \tag{3.1}$$

obtained from a linearization of the real nonlinear system $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t))$ using a first order Taylor expansion and then discretized using for example the Euler method. The quadratic cost function is defined as below

$$J(\boldsymbol{x}_i, \boldsymbol{u}_i) = \sum_{i=0}^{N-1} J_s(\boldsymbol{x}_i, \boldsymbol{u}_i) + J_N(\boldsymbol{x}_N) \tag{3.2}$$

where N is the optimization horizon, in this case finite, that represents how far we look into the future, and J_s, J_N are called respectively the *stage* and the *final* cost. The last term plays an important role in our optimization problem since it should define what happens from N to infinity, and hence should embed a meaningful information about future steps. The cost function J is usually composed by some quadratic terms that incentive the robot to reach a target position $\boldsymbol{x}_g$ maintaining there a final control inputs $\boldsymbol{u}_g$ or to follow a precomputed reference trajectory $(\boldsymbol{x}_i^{\text{ref}}, \boldsymbol{u}_i^{\text{ref}})$ for $i = 0, .., N$. In the first case, we have a regulation problem and J can be composed by the following stage and final costs

$$\begin{aligned} J_s(\boldsymbol{x}_i, \boldsymbol{u}_i) &= (\boldsymbol{x}_i - \boldsymbol{x}_g)^\top \boldsymbol{Q}(\boldsymbol{x}_i - \boldsymbol{x}_g) + (\boldsymbol{u}_i - \boldsymbol{u}_g)^\top \boldsymbol{R}(\boldsymbol{u}_i - \boldsymbol{u}_g) && (3.3) \\ J_N(\boldsymbol{x}_N) &= (\boldsymbol{x}_N - \boldsymbol{x}_g)^\top \boldsymbol{Q}_N(\boldsymbol{x}_N - \boldsymbol{x}_g). && (3.4) \end{aligned}$$

where $\boldsymbol{Q}$, $\boldsymbol{R}$, $\boldsymbol{Q}_f$ are positive-definite weights matrices. From now on, to simplify the notation, we will assume a robotics task where we need to stabilize the system around a free equilibrium point, i.e. $(\boldsymbol{x}_g, \boldsymbol{u}_g) = (\boldsymbol{0}, \boldsymbol{0})$, hence eqs. (3.3–3.4) can be simply rewritten as

$$\begin{aligned} J_s(\boldsymbol{x}_i, \boldsymbol{u}_i) &= \boldsymbol{x}_i^\top \boldsymbol{Q}\boldsymbol{x}_i + \boldsymbol{u}_i^\top \boldsymbol{R}\boldsymbol{u}_i \\ J_N(\boldsymbol{x}_N) &= \boldsymbol{x}_N^\top \boldsymbol{Q}_N \boldsymbol{x}_N. \end{aligned}$$

The *discrete* LQR problem can be defined as a minimization of the above cost function J, plus an equality constraint dictated by the system model (3.1). Formally, it can be expressed by the following optimization problem

$$\min_{\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{N-1}} \sum_{i=0}^{N-1} J_s(\boldsymbol{x}_i, \boldsymbol{u}_i) + J_N(\boldsymbol{x}_N) \tag{3.5}$$

subject to

$$\boldsymbol{x}_{i+1} - \boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{B}\boldsymbol{u}_i = 0, \qquad i = 0, \ldots, N-1 \tag{3.6}$$

that once solved, will give us back a *sequence* of optimal control inputs to drive our system. Different ways exist to solve this QP, and above all DP (Sec. 2.2) is one of the most common approaches. In Dynamic Programming, we seek to optimize our cost function backward in time starting from the known final cost $\boldsymbol{Q}_N$. For example, following the update rule of eq. (2.21), and defining $\boldsymbol{P}_N = \boldsymbol{Q}_N$, we can find in closed form the relationship between the value function at time N and its value at time $N-1$ as below

$$\begin{aligned} J_N\left(\boldsymbol{x}_N\right) &= \boldsymbol{x}_N^\top \boldsymbol{P}_N \boldsymbol{x}_N \\ J_{N-1}\left(\boldsymbol{x}_{N-1}\right) &= \boldsymbol{x}_{N-1}^\top \boldsymbol{Q} \boldsymbol{x}_{N-1} + \min_{\boldsymbol{u}_k}(\boldsymbol{u}_{N-1}^\top \boldsymbol{R} \boldsymbol{u}_{N-1} + \boldsymbol{x}_N^\top \boldsymbol{P}_N \boldsymbol{x}_N) \end{aligned} \tag{3.7}$$

where in the last equation the variable $\boldsymbol{x}_N$ can disappear substituting inside the linear model of the system $\boldsymbol{x}_N = \boldsymbol{A}\boldsymbol{x}_{N-1} + \boldsymbol{B}\boldsymbol{u}_{N-1}$. At the time $N-1$, since we opted for a quadratic cost function plus a linear model, we can calculate in closed form the optimal control inputs $\boldsymbol{u}_{N-1}$ putting the derivative of J_{N-1} over the control inputs equal to zero, obtaining the one step control law

$$\boldsymbol{u}_{N-1} = -\left(\boldsymbol{R}^{-1}\boldsymbol{B}^\top \boldsymbol{P}_N \boldsymbol{B}\right)^{-1} \boldsymbol{B}^\top \boldsymbol{P}_N \boldsymbol{A} \boldsymbol{x}_{N-1}$$

which, if substituted back in eq. (3.7), will give us a new cost matrix P_{N-1} equals to

$$\boldsymbol{P}_{N-1} = \boldsymbol{Q} + \boldsymbol{A}^\top(\boldsymbol{P}_N - \boldsymbol{P}_N\boldsymbol{B}(\boldsymbol{R} + \boldsymbol{B}^\top\boldsymbol{P}_N\boldsymbol{B})^{-1}\boldsymbol{B}^\top\boldsymbol{P}_N)\boldsymbol{A}. \tag{3.8}$$

Eq. (3.8) is called the *Riccati difference equation.* We can continue the iteration until the initial time 0 is reached, obtaining finally the optimal control inputs to be applied at the current robot state $\boldsymbol{x}_0$. An interesting feature of the DP method is that, taking an horizon N large enough, the matrix $\boldsymbol{P}_i$ in one of the backward steps will converge at some point to a fixed value $\boldsymbol{P}$, which is the solution of the *infinite* version of optimization problem considered. There, the single matrix $\boldsymbol{P}$ is usually found by solving the equation below

$$\boldsymbol{P} = \boldsymbol{Q} + \boldsymbol{A}^\top(\boldsymbol{P} - \boldsymbol{P}\boldsymbol{B}(\boldsymbol{R} + \boldsymbol{B}^\top\boldsymbol{P}\boldsymbol{B})^{-1}\boldsymbol{B}^\top\boldsymbol{P})\boldsymbol{A}. \tag{3.9}$$

called the *Discrete Time Algebraic Riccati equation* (DARE). This correspondence is important since it gives us the possibility to solve just *once* the DP recursion, offline and for a very large horizon, precomputing the fixed $\boldsymbol{P}_i \approx \boldsymbol{P}$ matrix and hence an optimal control law that can be emplyed for each possible initial state of our system, thus avoiding any control overhead during robot motion.

Examples of LQR in robotics can be found in [63], where it is used in a whole-body control setting for a two-wheeled inverted pendulum named Ascento. In [64] Machine Learning and LQR are combined together to find the optimal set of cost matrices $\boldsymbol{Q}$,$\boldsymbol{R}$, $\boldsymbol{Q}_N$ adopting Bayesian Optimization, which is an optimization method that leverages GP (Sec. 2.1.1) for reconstructing the unknown function to optimize. Finally, LQR can be even used for a tracking task, as done in [65] for a cart-pendulum robot. In this case, we should linearize the nonlinear model of the system around the desired trajectory and not anymore on a single equilibrium point. We will obtain in this case a linear *time-variant* system, and during the backward computation of DP the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ will not be anymore constant but will change along the horizon.

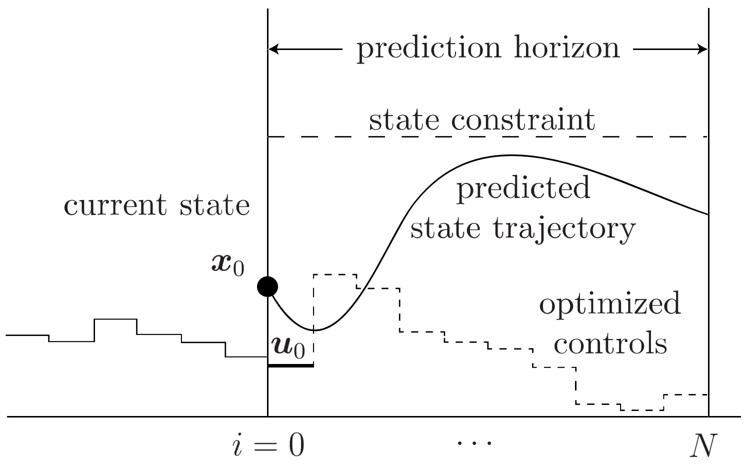

Figure 3.1. MPC receding horizon strategy. At each control step, the optimization problem is solved starting from the current state $\boldsymbol{x}_0$ and only the first control inputs $\boldsymbol{u}_0$ si applied to the system.

3.2 Model Predictive Control

Unfortunately, it is not always possible to apply the LQR control law to a robot. This can be due to a variety of reasons, for example, the required control law could exceed the motor's maximum torque, or other physical constraints such as the maximum angular velocity of a rotational joint could be violated during motion, thus making the prediction in eq. (3.6) not anymore consistent with reality. In all of these cases, we need to solve a different QP problem, where state and inputs constraints are actually enforced during optimization. The most famous controller that nowadays is able to deal with such cases is linear MPC [66]. It tries to solve at each control step k a constrained QP formulated as following

$$\mathrm{QP}(\boldsymbol{x}_0) = \min_{\boldsymbol{u}_0,\ldots,\boldsymbol{u}_{N-1}} \sum_{i=0}^{N-1} J(\boldsymbol{x}_i, \boldsymbol{u}_i)$$

subject to

$$\begin{aligned} \boldsymbol{x}_{i+1} - \boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{B}\boldsymbol{u}_i &= 0, & i &= 0,\ldots,N-1, \\ \boldsymbol{G}_i\boldsymbol{x}_i &\leq 0, & i &= 1,\ldots,N, \\ \boldsymbol{H}_i\boldsymbol{u}_i &\leq 0, & i &= 0,\ldots,N-1, \end{aligned} \tag{3.10}$$

where the last two inequalities are the desired state and input linear constraints that we would not violate, and where the cost function J, for example, is formulated as in eqs. (3.3–3.4) to solve a regulation task. Notice that in eq. (3.10) we use can actually get rid of the equality constraints dictated by the dynamical model. In fact, we can forward integrating the system, thus eliminating all the state variables $\boldsymbol{x}_i$. This approach is called the *single shooting* method.

MPC applies a receding horizon strategy in the control law (see Fig. 3.1), i.e. it solves the aforementioned constrained QP starting from the actual state of the robot $\boldsymbol{x}_0 = \boldsymbol{x}_k$, and then it applies only the *first* control inputs $\boldsymbol{u}_0$ to the system.

After that, it solves again the control problem from the successive robot state and so on. The need for such strategy derives primarily for obtaining a closed loop behaviour of the controller, since differently from the LQR problem (3.1), MPC computes the optimal state feedback *only* for the current state and it does not give us a general control law. To be solved, MPC needs specialized solvers which can be very computationally demanding. For this reason, the increase of the complexity of the problem is usually compensated by choosing a shorter prediction horizon N, but this can have serious drawbacks on the pure performance of the final controller, i.e. the closed loop system could be even unstable. To solve this issue, a common approach is to mix both LQR and MPC together, taking the best from the two control laws. For example, we could formulate the cost function J of the MPC as following

$$\begin{aligned} J_s(\boldsymbol{x}_i, \boldsymbol{u}_i) &= \boldsymbol{x}_i^\top \boldsymbol{Q} \boldsymbol{x}_i + \boldsymbol{u}_i^\top \boldsymbol{R} \boldsymbol{u}_i \\ J_N(\boldsymbol{x}_N) &= \boldsymbol{x}_N^\top \boldsymbol{P} \boldsymbol{x}_N. \end{aligned}$$

where we set the matrix $\boldsymbol{P}$ obtained by the LQR recursion as the *final cost* of our problem. As detailed previously, $\boldsymbol{P}$ embeds future information about the optimization problem, hence the shorter N is actually compensated by a prediction of the cost encountered in the future steps. This is usually an approximation of the considered constrained QP problem, since $\boldsymbol{P}$ does not contain any information about constraints, but in many cases if we are *near enough* the equilibrium point/reference trajectory, constraints could not be violated by even an unconstrained control law, thus making $\boldsymbol{P}$ the true final cost. In fact, violations can usually happen when we are far away from the regulation point since the controller could require a motion that would exceed the robot constraints. In the case of tracking the same reasoning applies, with an additional assumption that the reference trajectory is feasible for the robot, i.e. once the robot carefully reproduces the trajectory its constraints are not violated.

Although systems in robotics have a nonlinear model, linear MPC is still widely adopted in academia and in the industry along with a simplified model that describes the system along some reference trajectory or equilibrium point ([67, 68]), similar to what is done for LQR. Furthermore, in the case of model uncertainties many approaches exist in the context of linear MPC to obtain a robust variant of this controller, named Robust MPC (see [69, 70]).

Many times we do not have any reference trajectory to follow *a-priori*, or in the case of regulation we could be too far away from the equilibrium, hence invalidating the linearization procedure of the system. In this case, we can try to solve a different optimization problem that considers the nonlinearities of the system at full. In this case, we talk about Nonlinear MPC (NMPC). In NMPC we usually start from an initial guess of the solution ($\boldsymbol{x}_i^{\text{guess}}$, $\boldsymbol{u}_i^{\text{guess}}$) for $i = 0, .., N$, which can be for example a reference trajectory precomputed or a vector of zero elements, both for controls and states, meaning that we do not have any assumption about how to solve the problem. The NMPC is usually solved by applying the Sequential Quadratic Programming (SQP) method [71], which iteratively solves quadratic approximations of the following NonLinear Program (NLP)

$$\mathrm{NLP}(\boldsymbol{x}_0) = \min_{\boldsymbol{u}_0,\dots,\boldsymbol{u}_{N-1}} \sum_{i=0}^{N-1} J(\boldsymbol{x}_i, \boldsymbol{u}_i)$$

subject to

$$\begin{aligned} \boldsymbol{x}_{i+1} - \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) &= 0, \qquad i = 0\dots, N-1, \\ \boldsymbol{g}(\boldsymbol{x}_i) &\leq 0, \qquad i = 1, \dots, N, \\ \boldsymbol{h}(\boldsymbol{u}_i) &\leq 0, \qquad i = 0, \dots, N-1 \end{aligned} \tag{3.11}$$

until convergence is achieved. Noticed that in the NLP above, the first equality constraint is represented directly by the nonlinear dynamics of the system $\boldsymbol{f}$, while $\boldsymbol{g}$ and $\boldsymbol{h}$ are possible nonlinear inequality constraints. In order to solve it, in the SQP approach we sequentially approximated by QPs the above NLP starting from the available initial guess. We can formulate a generic single QP at the SQP iteration j as

$$\mathrm{QP}(\boldsymbol{x}_0, \boldsymbol{x}_j^{\text{guess}}, \boldsymbol{u}_j^{\text{guess}}) = \min_{\Delta\boldsymbol{u}_0,\dots,\Delta\boldsymbol{u}_{N-1}} \sum_{i=0}^{N-1} J(\Delta\boldsymbol{x}_i, \Delta\boldsymbol{u}_i)$$

subject to

$$\begin{aligned} \Delta\boldsymbol{x}_i = \boldsymbol{x}_i - \boldsymbol{x}_{j,i}^{\text{guess}}, &\qquad i = 0, \dots, N-1, \\ \Delta\boldsymbol{u}_i = \boldsymbol{u}_i - \boldsymbol{u}_{j,i}^{\text{guess}}, &\qquad i = 0, \dots, N-1, \\ \Delta\boldsymbol{x}_{i+1} - \boldsymbol{A}\Delta\boldsymbol{x}_i - \boldsymbol{B}\Delta\boldsymbol{u}_i = 0, &\qquad i = 0, \dots, N-1, \\ \boldsymbol{G}_i\Delta\boldsymbol{x}_i \leq 0, &\qquad i = 1, \dots, N, \\ \boldsymbol{H}_i\Delta\boldsymbol{u}_i \leq 0, &\qquad i = 0, \dots, N-1, \end{aligned} \tag{3.12}$$

where

$$\begin{aligned} \boldsymbol{A}_i &= \left.\frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{j,i}^{\text{guess}}, \boldsymbol{u}_{j,i}^{\text{guess}}} \\ \boldsymbol{B}_i &= \left.\frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{u}}\right|_{\boldsymbol{x}_{j,i}^{\text{guess}}, \boldsymbol{u}_{j,i}^{\text{guess}}} \\ \boldsymbol{G}_i &= \left.\frac{\partial \boldsymbol{g}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{j,i}^{\text{guess}}, \boldsymbol{u}_{j,i}^{\text{guess}}} \\ \boldsymbol{H}_i &= \left.\frac{\partial \boldsymbol{h}(\boldsymbol{u})}{\partial \boldsymbol{u}}\right|_{\boldsymbol{x}_{j,i}^{\text{guess}}, \boldsymbol{u}_{j,i}^{\text{guess}}}, \end{aligned}$$

are the linear approximation of the nonlinear dynamics $\boldsymbol{f}$ and the state and input constraints along with the provided initial guess. It should be noticed that, if no initial guess is provided, i.e. they are composed by zero elements, the two QP problems (3.10–3.12) are *equivalent* since the linearization is performed around a free equilibrium point. After the solution of (3.12), we can update the previous initial guess as following

$$(\boldsymbol{x}_{j+1}^{\text{guess}}, \boldsymbol{u}_{j+1}^{\text{guess}}) = (\boldsymbol{x}_j^{\text{guess}}, \boldsymbol{u}_j^{\text{guess}}) + (\Delta\boldsymbol{x}, \Delta\boldsymbol{u}) \tag{3.13}$$

and then continue to iterate until the cost J does not decrease anymore. Furthermore, in order to avoid an abrupt change of the initial guess during iteration, usually a constant $\alpha \leq 1$ is premultiplied to each QP solution and the initial guess update is performed as below

$$(\boldsymbol{x}_{j+1}^{\text{guess}}, \boldsymbol{u}_{j+1}^{\text{guess}}) = (\boldsymbol{x}_{j}^{\text{guess}}, \boldsymbol{u}_{j}^{\text{guess}}) + \alpha(\Delta\boldsymbol{x}, \Delta\boldsymbol{u}).$$

The full SQP algorithm to solve the NLP (3.11) is detailed in Table 4.

Algorithm 4 SQP algorithm

```
Require: initial state $\boldsymbol{x}_0$, initial guess ($\boldsymbol{x}_j^{\text{guess}}$, $\boldsymbol{u}_j^{\text{guess}}$), $j = 0$
1: While not converged:
2:   compute $\boldsymbol{A}_i$,$\boldsymbol{B}_i$,$\boldsymbol{G}_i$,$\boldsymbol{H}_i$ by a first order linearization over ($\boldsymbol{x}_k^{\text{guess}}$, $\boldsymbol{u}_k^{\text{guess}}$)
3:   solve QP($\boldsymbol{x}_0$, $\boldsymbol{x}_j^{\text{guess}}$, $\boldsymbol{u}_j^{\text{guess}}$)
4:   update the initial guess ($\boldsymbol{x}_{j+1}^{\text{guess}}$, $\boldsymbol{u}_{j+1}^{\text{guess}}$) = ($\boldsymbol{x}_j^{\text{guess}}$, $\boldsymbol{u}_j^{\text{guess}}$) + $\alpha$ ($\Delta\boldsymbol{x}$,$\Delta\boldsymbol{u}$)
5:   $j = j + 1$
```

Solving the NLP problem by applying the SQP method until convergence can be very time consuming. The speed of this computation usually depends strongly on the initial guess provided to the problem in two different ways: first avoiding the SQP algorithm to exit with an infeasible solution; second, it allows to take a full step with $\alpha = 1$ hence allowing for a fast convergence rate. In the specific context of SQP for NMPC, a very good initial guess for the discrete control time instant k can be constructed adopting the solution obtained at time $k - 1$ as following

$$\begin{aligned} \boldsymbol{x}_{k,i}^{\text{guess}} &= \boldsymbol{x}_{k-1,i+1}, & i &= 0, \dots, N-1, \\ \boldsymbol{u}_{k,i}^{\text{guess}} &= \boldsymbol{u}_{k-1,i+1}, & i &= 0, \dots, N-2, \\ \boldsymbol{x}_{k,N}^{\text{guess}} &= \boldsymbol{f}(\boldsymbol{x}_{k-1,N}, \boldsymbol{u}_{k-1,N-1}) \end{aligned} \tag{3.14}$$

where with $(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1})$ we denote the *previous* NMPC solution at time $k - 1$. Notice that in eq. (3.14) we only miss the last terms at time N and at time $N - 1$, respectively for the state and for the input. An easy solution to this problem, is just replicating the last control input at time $N - 2$, obtaining concurrently the terminal state by forward integration of the nonlinear dynamics. In this case, if the solution at time $k - 1$ is feasible for the problem, then the shifted solution is feasible (except for the last time step) with respect to the dynamic constraints. Additionally, if the guess for the time instant k obtained via the shifting procedure is sufficiently close to the exact solution of the NMPC problem, then we can just perform one iteration of SQP since we provide already an excellent approximation of the exact solution to the NMPC problem.

Other solutions exist in literature for initializing the SQP iteration. For example, an RL policy can be learned offline and queried at run-time to provide a good initial guess to the optimization problem, hence giving stronger assurance in the performance of the final control law. In fact, in a continuous state space we do not have any assurance of the performance of RL, which can return a very poor solution in some parts of the state-space. On the other hand, a poor initialization of the SQP algorithm can make the NMPC not run in real-time, hence using as bootstrap the RL can help to speed-up the solution of the NLP.

3.3 The Real Time Iteration scheme

Even if we smart initialize the initial guess, SQP introduces a high delay in the computation of the control law. In fact, only the first iteration of the algorithm contains a true estimate of the initial state of the problem $\boldsymbol{x}_k$, while, since time is evolving during computation, the following iterations will be based on *old* estimation that is not anymore valid. To solve this issue, the real-time iteration algorithm was introduced in [72]. It consists in performing only one iteration step of the SQP iterations using the shifting procedure in eq. (3.14), plus it divides the computation of the optimization problem in two different phases, called *preparation phase* and *feedback phase*. In the preparation phase, the shifting procedure is performed, and then the QP (3.12) is constructed performing the linearization of the nonlinear dynamics and constraints. This phase can be conducted just after completing the previous NMPC step, *without* waiting for the next sampling interval. Next, when the new state of the robot is available from sensors, in the feedback phase only *one* iteration of SQP is performed. A sketch of the algorithm can be found in Table 5.

Algorithm 5 RTI algorithm

```
preparation phase
Require: $(\boldsymbol{x}_k,\ \boldsymbol{u}_k)$ from the previous NMPC solution at time $k-1$
1:  shift initial guess according to eq. (3.14)
2:  compute $\boldsymbol{A}_i,\boldsymbol{B}_i,\boldsymbol{G}_i,\boldsymbol{H}_i$ by a first order linearization over $(\boldsymbol{x}_k^{\text{guess}},\ \boldsymbol{u}_k^{\text{guess}})$
3:  return QP$(\boldsymbol{x}_0,\ \boldsymbol{x}_k^{\text{guess}},\ \boldsymbol{u}_k^{\text{guess}})$

feedback phase
Require: initial state $\boldsymbol{x}_0 = \boldsymbol{x}_k$ obtained in the current sampling interval $k$
1:  compute $(\Delta\boldsymbol{x},\ \Delta\boldsymbol{u})$ solving QP$(\boldsymbol{x}_0,\ \boldsymbol{x}_k^{\text{guess}},\ \boldsymbol{u}_k^{\text{guess}})$
2:  update initial guess $(\boldsymbol{x}_k^{\text{guess}},\ \boldsymbol{u}_k^{\text{guess}}) = (\boldsymbol{x}_k^{\text{guess}},\ \boldsymbol{u}_k^{\text{guess}}) + \alpha(\Delta\boldsymbol{x},\Delta\boldsymbol{u}),\ \alpha = 1$
```

The separation into distinct phases reduces the delay to a minimum. In fact, first the construction of the QP (3.12) that in general can take some time does not introduce any delay in this formulation, since all the computation is performed before the control instant k. Second, thanks to the smart shifting solution and the single one SQP iteration, the feedback phase is in general very fast to compute, with a complexity that is *equal* to the one obtained in the linear MPC (3.2). In fact, remember that linear MPC and NMPC are equal if just one iteration is performed. Furthermore, the sampling time that can be achieved via RTI based NMPC increases from standard linear MPC by the time required for the preparation phase.

In literature two main open source libraries exist that provide an efficient implementation of RTI: the `ACADO Toolkit` [73] and `acados` [74]. They have the peculiarity of providing autogenerated C++ code that can be used for fast real-time applications (see [75, 76, 77]). In this thesis will use the first library for obtaining a control law for the Pendubot in Ch. 6.

3.4 Quadratic Program solution

Until now we have not yet discussed how to solve the QP generated by the linear MPC and by the SQP method at each iteration. The aim of this section is to provide to the reader one solution method from many, named the *active-set* algorithm, which is used as a backbone of all the optimization problems that will be formulated in the following chapters. Before starting its explanation, it is necessary to introduce the famous *Karush-Kuhn-Tucker* (KKT) optimality conditions and the concept of the *Lagrange multipliers*, which play a crucial role in numerical methods for optimal control. Let us assume a simplified version of the QP problem (3.10) with an horizon $N = 2$ and two inequalities constraints $h_1(u_1) \leq 0$, $h_2(u_2) \leq 0$ on the only two inputs that are available. Furthermore, remember that equality constraints dictated by the robot dynamics can actually be eliminated by the QP by forward integration of the system model, thus obtaining a QP that is dependant only on the control inputs variables. The KKT conditions state that if $\boldsymbol{u}^* = (u_1^*, u_2^*)$ is a local optimizer of the QP, then there exists the so called multiplier vector $\boldsymbol{\mu}^* = (\mu_1^*, \mu_2^*)$ such that

$$\begin{aligned} L(\boldsymbol{x}_0, \boldsymbol{u}^*, \boldsymbol{\mu}^*) = \nabla J\left(\boldsymbol{x}_0, \boldsymbol{u}^*\right) + \nabla h_1\left(u_1^*\right)\mu_1^* + \nabla h_2\left(u_2^*\right)\mu_2^* &= 0 \\ h_1\left(u_1^*\right) &\leq 0 \\ h_2\left(u_2^*\right) &\leq 0 \\ \boldsymbol{\mu}^* &\geq 0 \\ \mu_1^* h_1\left(u_1^*\right) &= 0 \\ \mu_2^* h_2\left(u_2^*\right) &= 0. \end{aligned} \tag{3.15}$$

where $L(\boldsymbol{x}_0, \boldsymbol{u}^*, \boldsymbol{\mu}^*)$ is called the lagrangian equation. Notice that eqs. (3.15) are a *simplified* version of the more general KKT conditions that can be found in [78], since here we are just considering inequalities constraints.

Motivating the KKT conditions is fairly simple. Rewriting the inequalities constraints as $\boldsymbol{d} = (h_1, h_2)$, they states that at the solution $\boldsymbol{u}^*$ the gradient of the cost function J is balanced by the gradient of $\boldsymbol{d}$ multiplied the lagrange multipliers $\boldsymbol{\mu}$. This can happen in three different ways. First, we can have $\boldsymbol{\mu}^* = 0$ obtaining

$$\begin{aligned} \nabla J\left(\boldsymbol{x}_0, \boldsymbol{u}^*\right) &= 0 \\ \nabla \boldsymbol{d}\left(\boldsymbol{u}^*\right) &< 0 \end{aligned} \tag{3.16}$$

where the inequalities $\boldsymbol{d}$ are not *active* and removing them does not change the global solution of the problem. Second, it can happen that $\boldsymbol{\mu}^* > 0$ and thus

$$\begin{aligned} \nabla J\left(\boldsymbol{x}_0, \boldsymbol{u}^*\right) &\neq 0 \\ \nabla \boldsymbol{d}\left(\boldsymbol{u}^*\right) &= 0 \end{aligned} \tag{3.17}$$

hence the solution is at the border of both the constraints $\boldsymbol{d}$ which are considered active. Finally, the third possibility is that one constraint i is active ($\mu_i^* > 0$), while the other does not play any role in the optimization problem. These three different possibilities are drawn in Fig. 3.2.

In order to solve the constrained QP, in the *active-set* method we check iteratively all the three possibilities above. First, we verify if the unconstrained solution of the

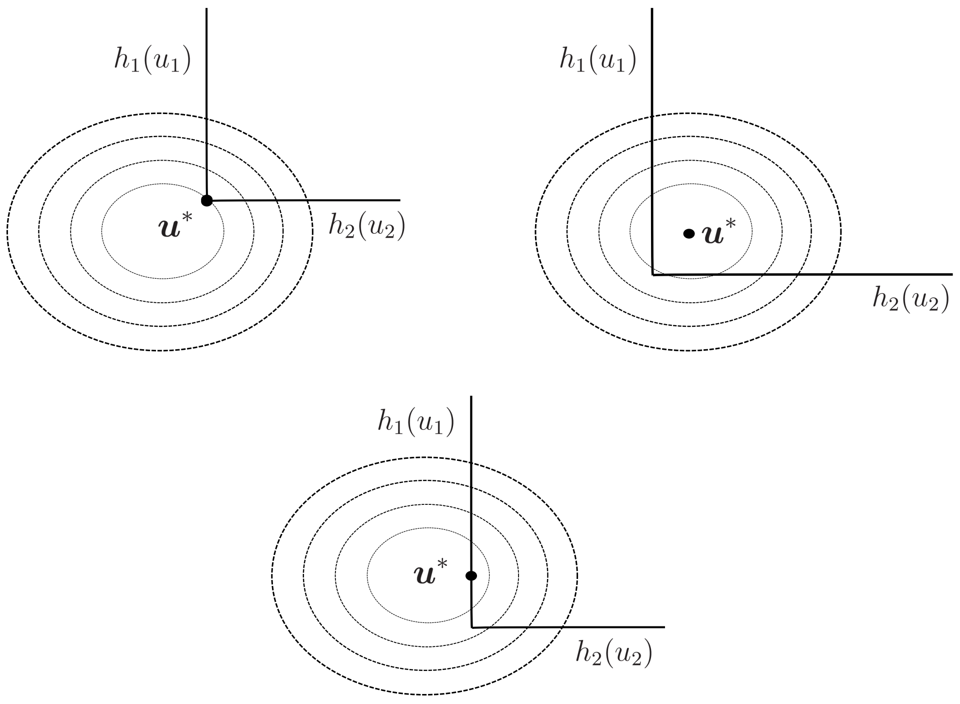

Figure 3.2. The three different scenarios in which eqs. (3.15) are satisfied. In the top-left both constraints are active at the solution, while in the top-right their elimination does not change the optimum of the problem. Finally, in the bottom only h_1 is active.

QP satisfies eqs. (3.16). This can be done taking the gradient of J over the decision variable $\boldsymbol{u}$ or by DP recursion, obtaining the global unconstrained minimum $\boldsymbol{u}^*$. If the KKT equations are respected, we have actually found the global solution and we stop. If not, we start by choosing an initial guess for the solution $\boldsymbol{u}^*$ along with some active constraints, e.g. $h_1(u_1^*) = 0$, and then we optimize a simpler unconstrained QP neglecting all the other inequalities. Notice that in our two-step horizon example, this means optimizing only the second input u_2 that remained free. If still eqs. (3.15) do not hold, we can change the active constraint and continue until convergence. Iterating this procedure in fact, we can actually find which constraints are active and which are not, and at the end we will obtain the constrained global optimum of the QP. As can be imagined, this methodology can be very time consuming, since it depends strongly on the number of constraints that are present in our problem. Still, this computation can be made faster by exploiting another warm-start procedure. In fact, the set of the active constraints from the previous QP solution can be easily exploited and used as an initial guess for the current iteration.

For one of the most efficient active-set solvers the reader can refer to `QPOASES` [79] which is one of the solution methods available in the library `ACADO`.

3.5 Chapter Summary

This chapter gave an overview on Optimal Control. We started by explaining LQR, which is a linear controller very popular in robotics especially in underactuated robots, such as the Pendubot, where it can be used for balancing the system around an equilibrium point. We then explained the basics of MPC, introducing the need for constraints satisfaction in a real system, and we focused on the computational complexity of both its linear and nonlinear version. In the second case, we introduced an iterative method called SQP which will be used in Ch. 5 directly for planning and not for control, and then we discussed an efficient version of the NMPC based on the Real Time Iteration scheme. Finally, we concluded this chapter explaining how the QP generated by all these constrained controllers can be solved, presenting an overview of the *active-set* method.

Chapter 4

An online learning procedure for feedback linearization control

4.1 Motivation and Contribution

As stated in the introduction, the knowledge of accurate dynamic models is of paramount importance for several robotic applications. It is necessary, in fact, for designing control laws with superior performances [80], during robot interactions with the environment (for example when implementing strategies for the sensorless detection, isolation and reaction to unexpected collisions [81]), or when regulating force or imposing a desired impedance control at the contact [82] is required. In order to retrieve an estimation of the dynamic model, regression techniques are widely employed [83, 84]. These techniques are hinged on a well-known property: the linear dependence of the robot dynamic equations in terms of a set of ρ *dynamic coefficients* [85], (also denoted in the literature as *base parameters*) [86], which are linear combinations of the *dynamic parameters* of the links composing the robot (masses, centers of gravity and inertia tensors).

In order to retrieve a reliable estimation of the dynamic model, a series of exciting trajectories are typically commanded to the robot, and the joint positions and torques are recorded during motion. It is eventually possible to obtain a numerical estimation of the dynamic coefficients by exploiting the filtered joint torques and the filtered joint positions, velocities and accelerations obtained by numerical derivation [86]. Typically, this whole procedure, whose output is the estimation of the dynamic model of the robot under study, is performed offline. Therefore, in case of changes in the structural parameters of the robot, the identification procedure has to be carried out again from scratch. This is the case, for instance, when unknown payloads are attached to the end-effector of a manipulator: in this regard, a method to update the dynamic model has been presented recently [87], but it requires an initial phase of setup for updating the dynamic model parameters. Especially when collision detection and reaction strategies are adopted during motion, it is essential to have an adaptable and reliable estimation of the dynamic model, employed to properly implement those algorithms [81].

In the last years, to overcome the limitations imposed by the aforementioned approaches, a new set of techniques has emerged relying on the employment of regression techniques to face the problem of robotics model learning. As stated previously, in literature there exist two different way to tackle this issue, namely *direct model* [88], [89], [90], [91] and *inverse model* learning [92],[93],[94]. In regards to the first class of methods in [88], the authors propose to learn the transition probability model, while in [90] the authors try to reconstruct the system nonlinear dynamics with a GP in order to improve approximate linearization of the system around an operating point. In [91] the authors utilize the regressor as a predictive model for a nonlinear MPC. In our approach, we make use of a linear MPC in the control scheme, but we design a procedure for learning the inverse dynamics of the system. In the context of inverse model learning, in [95] the authors describe a way to learn a computed torque controller, employing the torque measurements to build their training dataset. In our work, we learn the unmodeled dynamics to improve the feedback linearization process without the use of any joint torque measurements, which are known to be noisy.

In this chapter, we propose a method to reconstruct dynamic model uncertainties and parameter variations by means of an online algorithm based on GP regression, which we described previously in Sec. 2.1.1. Having an *a-priori* estimation of the dynamic model of a robot, we show that it is possible to improve the model by exploiting only joint position measures, without the need for any joint torque data. Indeed, torque measurements are usually affected by a high level of noise, typically higher than the noise added to the measures coming from the encoders which return joint positions [80]: thus, the employed algorithm is able to obtain a reliable estimate of dynamic uncertainties.

This chapter is organized as follows. Sec. 4.2 introduces the dynamic model of fully actuated robots, highlighting the problem deriving from a not exact Feedback Linearization (FL) controller. In Sec. 4.3 the learning procedure is presented, with a description of the data collection procedure and of the proposed control architecture. In Sec. 4.4 simulation results for a 7-dofs LBR robots are shown, and finally in Sec. 4.5 a summary of the chapter is presented along with possible future works.

4.2 Problem Formulation

For a robot with n degree of freedom and n actuators, its dynamics can be written as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau}, \tag{4.1}$$

in which $\boldsymbol{q}$ is the n-dimensional configuration vector, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$ are, respectively, the joint velocities and accelerations; $\boldsymbol{M}$ is the inertia matrix and $\boldsymbol{n}$ is a vector obtained by the sum of the Coriolis and centrifugal forces and the gravity vector. Supposing that the robot is fully actuated, it is possible to design a FL controller [80], providing a control input $\boldsymbol{\tau}_{\mathrm{FL}}$ that cancels the nonlinear dynamics components of the model in eq. (4.1), as

$$\boldsymbol{\tau}_{\mathrm{FL}} = \boldsymbol{M}(\boldsymbol{q})\boldsymbol{u} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}), \tag{4.2}$$

where $\boldsymbol{u}$ represents the desired joint accelerations vector. In principle, given a perfect knowledge of the system dynamic model and applying the FL controller (4.2) we would get a linear system of double integrators

$$\ddot{\boldsymbol{q}} = \boldsymbol{u}, \tag{4.3}$$

but in reality this is typically not true due to unmodeled dynamics and uncertainties in the system parameters. If we explicitly account for these uncertainties in the model, we have therefore

$$\boldsymbol{M} = \hat{\boldsymbol{M}} + \Delta\boldsymbol{M} \qquad \boldsymbol{n} = \hat{\boldsymbol{n}} + \Delta\boldsymbol{n}. \tag{4.4}$$

where $\hat{\boldsymbol{M}}$ and $\hat{\boldsymbol{n}}$ are nominal quantities (for instance, previously estimated or given by the manufacturer of the robot), $\Delta\boldsymbol{M}$ and $\Delta\boldsymbol{n}$ are increments characterizing the uncertainties. Thus, $\hat{\boldsymbol{M}}$ and $\hat{\boldsymbol{n}}$ incorporate our *a-priori* knowledge of the system that we can exploit in a model-based control law.

If we apply the nominal FL control on the real system of eq. (4.1), that is

$$\hat{\boldsymbol{\tau}}_{\mathrm{FL}} = \hat{\boldsymbol{M}}\boldsymbol{u} + \hat{\boldsymbol{n}}, \tag{4.5}$$

considering eqs. (4.4), we obtain

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}\hat{\boldsymbol{M}}\boldsymbol{u} - \boldsymbol{M}^{-1}\Delta\boldsymbol{n} = \boldsymbol{u} + \boldsymbol{\delta}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \tag{4.6}$$

where $\boldsymbol{\delta}$ accounts for all the unmodeled dynamic and uncertainty terms. The presence of these model perturbations affects considerably the control problem, requiring usually the use of a high gain controller to track some reference trajectories which can even destabilize the system. Here instead, we want to retrieve an online estimate of $\boldsymbol{\delta}$ in order to directly eliminate the remaining unknown uncertainties.

4.3 The Proposed Approach

In this section, we describe the proposed online learning approach that is able to counteract these uncertainties in the dynamic model, composed by a dataset reconstruction method (Secs. 4.3.1–4.3.2), and by a linear but effective control law (Sec. 4.3.3). At its core there is a *learning process* which continuously updates a regressor $\boldsymbol{\varepsilon}$ estimate of the perturbation term $\boldsymbol{\delta}$ in (4.6) from position measurements during robot motion.

4.3.1 Dataset collection procedure

At first, we suppose that it is possible to drive the manipulator by commanding joint torques, that the current robot state is $\boldsymbol{x}_k = (\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k)$ and that we want to reach a desired target state $\boldsymbol{x}^{\mathrm{d}}_{k+1}$. We compute the input acceleration $\boldsymbol{u}_k$ that should drive the robot to the desired state supposing a perfect FL controller, i.e. by imposing the joint torque $\hat{\boldsymbol{\tau}}_{\mathrm{FL}}$ from eq. (4.5)

$$\boldsymbol{M}\ddot{\boldsymbol{q}}_k + \boldsymbol{n} = \hat{\boldsymbol{\tau}}_{\mathrm{FL},k} = \hat{\boldsymbol{M}}\boldsymbol{u}_k + \hat{\boldsymbol{n}}. \tag{4.7}$$

where $\boldsymbol{u}_k$ is chosen accordingly to reach $\boldsymbol{x}^{\mathrm{d}}_{k+1}$, but due to the effect of unmodeled dynamics the robot reaches a different state $\boldsymbol{x}_{k+1} = (\boldsymbol{q}_{k+1}, \dot{\boldsymbol{q}}_{k+1})$. At this point, from eq. (4.7), it is possible to compute the perturbation term $\boldsymbol{\delta}$ that depends only on the system state at time $\boldsymbol{x}_k$, the desired and actual system accelerations $\boldsymbol{u}, \ddot{\boldsymbol{q}}_k$ as

$$\ddot{\boldsymbol{q}}_k - \boldsymbol{u} = -\hat{\boldsymbol{M}}^{-1}(\Delta \boldsymbol{M} \ddot{\boldsymbol{q}}_k + \Delta \boldsymbol{n}). \tag{4.8}$$

In eq. (4.8) the term $\ddot{\boldsymbol{q}}_k$ cannot be known in advance and has to be computed *a posteriori* using the information about the states $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k+1}$. To this aim, we can utilize the notion of Controllability Gramian by calculating the true joints acceleration $\ddot{\boldsymbol{q}}_{g,k}$ that would have brought the perfectly feedback linearized system in the state $\boldsymbol{x}_{k+1}$ in the first place. In section 4.3.2 we will show that under certain conditions $\ddot{\boldsymbol{q}}_{g,k} = \ddot{\boldsymbol{q}}_k$.

Our learning framework is based on a torque controller to command the robot while collecting the data for estimating the model. By introducing a function regressor $\boldsymbol{\varepsilon}(\cdot)$ in the FL control law, we show that with our method it's possible to progressively (and thus online) improve the control performances exploiting all the data about the unknown dynamics acquired while commanding the robot. Therefore, we introduce the new FL control input $\boldsymbol{\tau}_{\mathrm{FL},k}$ where from the commanded high level acceleration $\boldsymbol{u}_k$ we subtract the regressor prediction $\boldsymbol{\varepsilon}(\cdot)$

$$\boldsymbol{\tau}_{\mathrm{FL},k} = \hat{\boldsymbol{M}}(\boldsymbol{u}_k - \boldsymbol{\varepsilon}_k) + \hat{\boldsymbol{n}}. \tag{4.9}$$

in order to cancel at best the perturbation $\boldsymbol{\delta}$. If we repeat this procedure for several states, we can incrementally construct a dataset $\mathcal{D} = \{(\boldsymbol{X}_i, \boldsymbol{Y}_i) \,|i = 1, \dots, n_d\}$, where n_d is the number of elements in our dataset. For each sample i, we have

$$\boldsymbol{X}_i = (\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i, \ddot{\boldsymbol{q}}_{g,i}); \quad \boldsymbol{Y}_i = \ddot{\boldsymbol{q}}_{g,i} - \boldsymbol{u}_i - \boldsymbol{\varepsilon}_i$$

where $\boldsymbol{\varepsilon}_i$ appear in order to preserve the correctness of the dataset since at each time step the robot moves under the modified control law of eq. (4.9). At this stage, it is important to point out that only the knowledge of states $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k+1}$ and of the desired acceleration $\boldsymbol{u}_i$ are needed while no torque information is necessary to build the dataset $\mathcal{D}$. Additionally, notice that in the published version of this work [18] we used a modified eq. (4.9), where the regressor $\boldsymbol{\varepsilon}$ is specified at the torque level and does not represent an acceleration. There is no practical difference in these two versions, and here the acceleration description is chosen only to ease the linking between Ch. 4 and Ch. 5.

When, a time k, we want to predict the compensation that is required to cancel out the unmodeled dynamics, the regressor inputs will be

$$\boldsymbol{\varepsilon}_k(\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \boldsymbol{u}_k)$$

where $\boldsymbol{u}_k$ is the desired joint acceleration obtained by a high level controller. Unfortunately, due to the prediction error of our regressor, small errors will be always committed that will be eventually reduced over time by increasing the dataset size.

4.3.2 Controllability Gramian

In order to reconstruct the unknown part of the dynamic model of the robot, the proposed estimation procedure requires the estimation of the real joint accelerations $\ddot{\boldsymbol{q}}_k$. For this purpose, we employ the concept of the Controllability Gramian starting from a generic continuous linear system

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \tag{4.10}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) \tag{4.11}$$

where $\boldsymbol{x}$ is the state vector, $\boldsymbol{A}$ is the state matrix, $\boldsymbol{B}$ is the input matrix, $\boldsymbol{u}$ is the input vector, $\boldsymbol{y}$ is the output vector and $\boldsymbol{C}$ is the output matrix, representing in this case the perfect feedback linearized system of eq. (4.3).

In this work, the robot state consists of joint positions and joint velocities, and in particular, for a n-DoFs robot, the state is $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$. Under the hypothesis of a perfectly feedback linearized system, it is possible to represent it as a chain of integrators, separating the whole system (4.11) into n independent subsystems of dimension 2. Therefore after the FL law, a single joint j, will have the following discrete state-space representation

$$\begin{aligned} \boldsymbol{x}^j_{k+1} &= \begin{pmatrix} 1 & T_s \\ 0 & 1 \end{pmatrix} \boldsymbol{x}^j_k + \begin{pmatrix} 0.5\ T_s^2 \\ T_s \end{pmatrix} \boldsymbol{u}^j_k \\ y^j_k &= \begin{pmatrix} 1 & 0 \end{pmatrix} \boldsymbol{x}^j_k. \end{aligned} \tag{4.12}$$

If the sampling time T_s is sufficiently small, the discrete system (4.12) approximates well its continuous counterpart (4.11). This system (4.12) is controllable since

$$\text{rank}(\boldsymbol{B}\ \boldsymbol{A}\boldsymbol{B}) = \text{rank} \begin{pmatrix} 0.5Ts^2 & 1.5Ts^2 \\ Ts & Ts \end{pmatrix} = 2,$$

and therefore it is possible to define the discrete Controllability Gramian as

$$\boldsymbol{W}(k-1) = \sum_{m=0}^{k-1} \boldsymbol{A}^m \boldsymbol{B}\boldsymbol{B}^T (\boldsymbol{A}^T)^m.$$

from which it is possible to retrieve the *minimum energy* input that drives the system from an initial state $\boldsymbol{x}_k$ to a final state $\boldsymbol{x}_{k+1}$ in m steps, as

$$u_k = -\boldsymbol{B}^T (\boldsymbol{A}^T)^{m-k} \boldsymbol{W}^{-1}(m)[(\boldsymbol{A}^T)^m \boldsymbol{x}_k - \boldsymbol{x}_{k+1}] \qquad k = 0, \ldots, m-1.$$

Since the systems consists in a chain of two integrators, only two steps are required to obtain $\ddot{q}_g$

$$\ddot{q}_g = u_0 + u_1 = (\boldsymbol{B}^T \boldsymbol{A}^T + \boldsymbol{B}^T) \boldsymbol{W}^{-1}(1)(\boldsymbol{A}^T \boldsymbol{x}_k - \boldsymbol{x}_{k+1}).$$

without using any numerical differentiation tools.

Finally, to summarize the whole process: in the case of perfect modeling, the FL would be exact and the system would act as a double integrator. Since the nominal model presents a mismatch with respect to the real one, the controlled system will behave in a different way, performing a diverse motion. This unexpected motion can be interpreted as if the feedback linearization was correct while the robot was driven by another unknown acceleration reference. Following this interpretation, it is always possible to use the Controllability Gramian on a double integrator in order to estimate this new reference acceleration, that will be used for the construction of the dataset. For this reason, the acceleration estimated by the Controllability Gramian is correct and it is independent from the current knowledge about the system's complete model.

4.3.3 Control Architecture

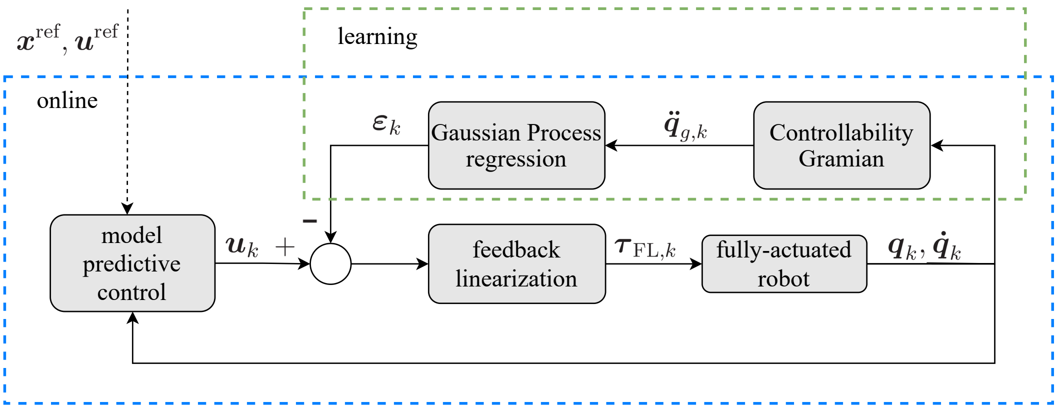
$\boldsymbol{x}^{\text{ref}}, \boldsymbol{u}^{\text{ref}}$
learning
online
$\boldsymbol{\varepsilon}_k$
Gaussian Process regression
$\ddot{\boldsymbol{q}}_{g,k}$
Controllability Gramian
−
model predictive control
$\boldsymbol{u}_k$ +
feedback linearization
$\boldsymbol{\tau}_{\text{FL},k}$
fully-actuated robot
$\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k$

Figure 4.1. Block diagram of the proposed algorithm. Solid signal lines represent data that are used at each time step, whereas the dashed line represents the desired trajectory computed offline.

In this section we describe in detail the control algorithm (see Fig. 4.1). Our method allows for robot trajectory tracking even if we have only a partial knowledge of its dynamic model. In order to achieve perfect FL, we fit online the obtained datapoints in $\mathcal{D}$ using GP regression. In this work we employ only the mean prediction value of eq. (2.13), represented here as $\boldsymbol{\varepsilon}$. Since no assumption has been made on its structure, a squared exponential kernel is employed (2.12) to define the covariance matrix. In addition, we set conditionally independent GPs for each joint of the manipulator.

In order to avoid unfeasible control actions or unfeasible reached states (i.e., out of known mechanical ranges), we generate the desired joint accelerations $\boldsymbol{u}_k$ through a linear MPC (see Sec. 3.2), which is able to provide smooth joint acceleration signals while satisfying state and input constraints. It should be noticed that we are not hinged with this particular choice, and other control laws could be implemented as well. For example, if we do not care about constraints in general, a simpler PD control law or the LQR (Sec. 3.1) could be used as well.

Under the assumption of perfect FL, the prediction model of the robot inside the MPC can be represented as a set of n independent linear double integrators with the discrete model described in eq. (4.12). Suppose now to have precomputed a reference trajectory $(\boldsymbol{x}^{\text{ref}}(t),\, \boldsymbol{u}^{\text{ref}}(t))$ that we can discretize by a sampling time interval T_s obtaining its counterpart $(\boldsymbol{x}_k^{\text{ref}},\, \boldsymbol{u}_k^{\text{ref}})$ for $k = 0, .., T/T_s$. The optimal control problem at the generic control time k can be written as

$$\min_{\boldsymbol{u}_0,\dots,\boldsymbol{u}_{N-1}} \sum_{i=0}^{N-1} J_s(\boldsymbol{x}_i,\boldsymbol{u}_i) + J_N(\boldsymbol{x}_N)$$

subject to

$$\begin{aligned} \boldsymbol{x}_{i+1} - \boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{B}\boldsymbol{u}_i &= 0, \qquad i = 0,\dots,N-1, \\ \boldsymbol{g}(\boldsymbol{x}_i) &\leq 0, \qquad i = 1,\dots,N, \\ \boldsymbol{h}(\boldsymbol{u}_i) &\leq 0, \qquad i = 0,\dots,N-1, \end{aligned}$$

with $\boldsymbol{x}_0 = \boldsymbol{x}_k$ the initial state of the robot at time k and $\boldsymbol{g}(\cdot)$ and $\boldsymbol{h}(\cdot)$ the desired state and input constraints. In this work we define the cost function as the sum of a stage cost J_s and a terminal cost J_N which penalize the deviation from the reference trajectory and smooth the control inputs

$$\begin{aligned} J_s(\boldsymbol{x}_i,\boldsymbol{u}_i) &= (\boldsymbol{x}_i^{\text{ref}} - \boldsymbol{x}_i)^\top \boldsymbol{Q}(\boldsymbol{x}_i^{\text{ref}} - \boldsymbol{x}_i) + (\boldsymbol{u}_i - \boldsymbol{u}_{i-1})^\top \boldsymbol{R}(\boldsymbol{u}_i - \boldsymbol{u}_{i-1}), \\ J_N(\boldsymbol{x}_N) &= (\boldsymbol{x}_N^{\text{ref}} - \boldsymbol{x}_N)^\top \boldsymbol{Q}_N(\boldsymbol{x}_N^{\text{ref}} - \boldsymbol{x}_N) \end{aligned} \tag{4.13}$$

where $\boldsymbol{Q}$, $\boldsymbol{Q}_N$, and $\boldsymbol{R}$ are the usual positive-definite, symmetric matrices of weights. The feasibility of the commanded acceleration $\boldsymbol{u}_k$ is obtained by imposing a set of state and control constraints $\boldsymbol{g}(\cdot)$ and $\boldsymbol{h}(\cdot)$ in our optimization problem

$$\begin{aligned} \boldsymbol{q}_m &\leq \boldsymbol{q} \leq \boldsymbol{q}_M \\ \dot{\boldsymbol{q}}_m &\leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{q}}_M \\ \ddot{\boldsymbol{q}}_m &\leq \boldsymbol{u} \leq \ddot{\boldsymbol{q}}_M \end{aligned}$$

where $\boldsymbol{q}_m, \dot{\boldsymbol{q}}_m, \ddot{\boldsymbol{q}}_m$ and $\boldsymbol{q}_M, \dot{\boldsymbol{q}}_M, \ddot{\boldsymbol{q}}_M$ are the lower and upper bounds for, respectively, joint positions, velocities and accelerations. The MPC constraints do not provide a strict safety guarantee at the beginning of the learning process, since the model in eq. (4.12) does not represent exactly the real system. Over time the online learning strategy improves the correction of the unmodeled dynamics resulting in a better correspondence between the MPC internal model and the real robot. Therefore, once the GP prediction error will became negligible, the optimal control input from the MPC will assure constraint satisfaction for the real manipulator as well.

The choice of the optimization horizon N is crucial for a real-time control law. In this work, we have not investigated this requirement, but in order to take a realistic scenario we have chosen $N = 20$ since the model is linear and the GP prediction is only computed *once* for each time step. The constrained optimization problem was solved in MATLAB using the `mpc` function and its default *active-set* solver, which we described in general in Sec. 3.4.

4.4 Simulation results

In this section we report the results of simulations performed applying the proposed method on a KUKA LBR iiwa 7 R800 manipulator performing a trajectory tracking task (Fig. 4.2). We analyze the tracking capabilities of the simulated robot, whose

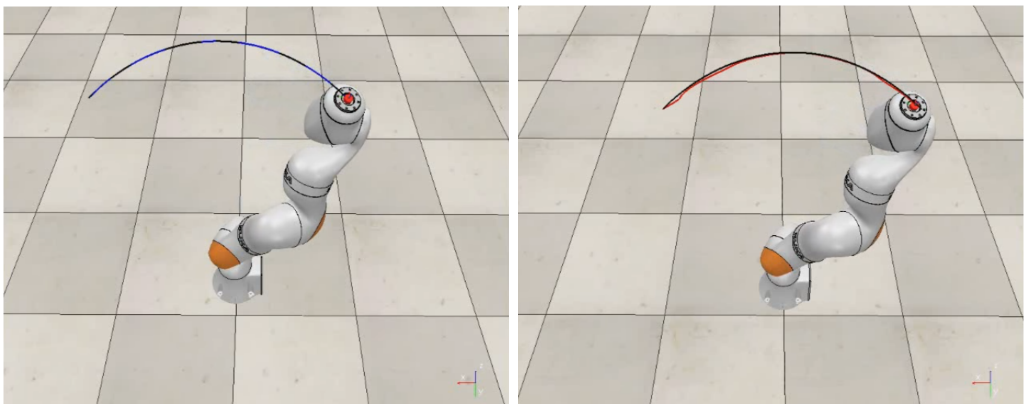

Figure 4.2. A KUKA LBR iiwa realizing a trajectory tracking task with the nominal controller (right, red) and with the proposed learning method (left, blue). A video of the simulation can be found at the following link `https://www.youtu.be/PfEt2G9MpHU`.

dynamic parameters are the ones reported in [96, 85] (originally identified for the academic version of the iiwa robot, namely the LightWeight Robot (LWR) 4+, which shares with the iiwa the same kinematics) with a deviation from the nominal parameters (around 20 % of their value), comparing the performances with and without the GP correction. In particular, we increase the nominal values of masses and CoM positions, modifying the barycentric inertias accordingly. Additionally, in the simulation we employ a friction model, that in general is considered difficult to estimate, with a nonlinear and non-smooth behavior (i.e., viscous and Coulomb friction with a Stribeck effect) [97] that was not considered in the nominal model of the system. Furthermore, the discretization time is set at $T_s = 1$ ms, while the high-level MPC and the dataset acquisition run at 200 Hz. The cost function matrices in (4.13) are diagonal, and equal to $\boldsymbol{R} = \boldsymbol{I}_n$, while $\boldsymbol{Q} = \boldsymbol{Q}_N = 2\boldsymbol{I}_{2n}$, where $\boldsymbol{I}_n$, $\boldsymbol{I}_{2n}$ are identity matrices of respectively n and $2n$ diagonal elements, with $n = 7$ the number of joints.

The simulation task consists in following a given reference trajectory in the joint space, both for joint positions and velocities. In particular, the first six joints should follow a sinusoidal path while the last one should remain at rest. Fig. 4.3 reports the joint angular position errors during the tracking task. The blue curves show the absolute value of the error when the proposed learning scheme of Fig. 4.1 is adopted. The red curves show, instead, the absolute values of the errors when only the nominal model (*a-priori* knowledge) is employed to compute the driving torque. It is evident that our learning scheme improves the quality of the tracking capabilities of the controller. In fact, already during the first moments there is an evident increment of the tracking performances in the joints 1, 4, 5, 6 and 7. This is achieved thanks to the high frequency of the data collection procedure employed, which permits a more accurate prediction of the disturbance since in this case from points that are *near enough* arise a similar value of $\boldsymbol{\delta}$.

The tracking error in the joint space has a direct effect on the cartesian error as well. Indeed, even small errors of the angular positions of the joints may dramatically deviate the end-effector position from the desired path. Fig. 4.4 reports the cartesian error at the end-effector level when the GP correction is active (blue curves) and when it is not (red curves). Even in this case, the use of the learning routines improves the quality of the tracking controller.

Additionally, in Fig. 4.5 is plotted a comparison between the learned regressor $\boldsymbol{\varepsilon}$ and the true mismatch $\boldsymbol{\delta}$ acting on the simulated system, divided by their contribution on each single joint. There, it can be seen that thanks to the high frequency of the dataset acquisition/prediction and the absence of additional noise, the GP is able to reconstruct the true signal accurately. Finally in Fig. 4.6 we show that, with our framework, the reference accelerations computed by the MPC and the measured accelerations of the robot converge thanks to the learned correction, meaning that the double integrator model used inside the controller becomes exact thanks to $\boldsymbol{\varepsilon}$.

4.5 Chapter Summary

This chapter presented a new approach for online learning an exact feedback linearization controller for a manipulator under model perturbations. The proposed method is composed by a dataset collection procedure that exploits the concept of Controllability Gramian and Gaussian Processes, and by a controller cascade composed by a Linear MPC that computes the commanding joint torques according to a desired trajectory plus a learned correction term that is able to counteract model uncertainties. It should be noticed that the computational time required for a prediction using GP is strictly dependant on the size of the dataset, but in this work we did not investigate this particular aspect since the GP prediction was not used inside the MPC formulation and it was computed only once for each control step. We showed that the proposed approach attained lower tracking error with respect to a nominal controller without any online correction, requiring only the knowledge of joint positions and velocities and without the need for any torque measurements.

Possible extensions of this algorithm will be the implementation of the proposed method on a real manipulator and an analysis of the effect of the learning transient for constraints satisfaction, which within this framework cannot be strictly assured in the initial moments. Another possibility could be to exploit the variance information of the GP regressor (2.14) applying the feedback linearization law at the cartesian level, exploiting the redundancy of the KUKA LBR iiwa to obtain equivalents joints motion with lower variance, i.e. with joints positions and velocities $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ that are most similar to the previously collected data. Furthermore, in this work we have considered the KUKA robot as a rigid manipulator, but in reality it has some flexibility in its mechanical structure between the joints and the motors. In that case, learning is complicated since most of the time we do not have encoders for both of them, hence a correct dataset $\mathcal{D}$ is difficult to obtain. A solution could be to describe the system using the joints variables up to the fourth order $\boldsymbol{q}^{[4]}$ [98], thus eliminating the need of using the motor variables to obtain a unique representation of the system.

In the next chapter, it will be presented the generalization of this approach for underactuated robots, which differently from full-actuated manipulators cannot undergo over a full feedback linearization procedure but still can be partially linearized.

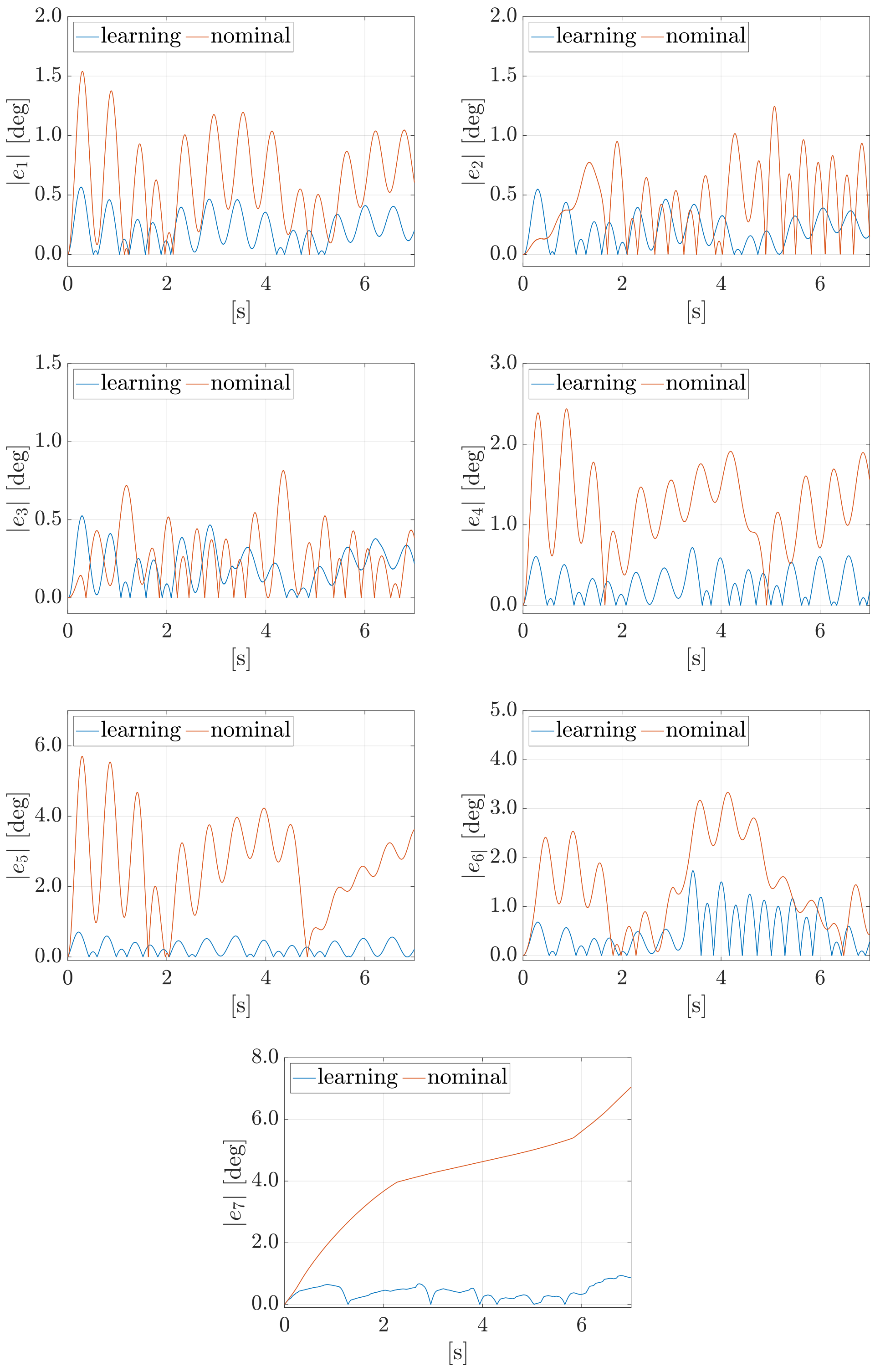

Figure 4.3. Joints position errors comparison: in red with the nominal model, in blue with learning enabled.

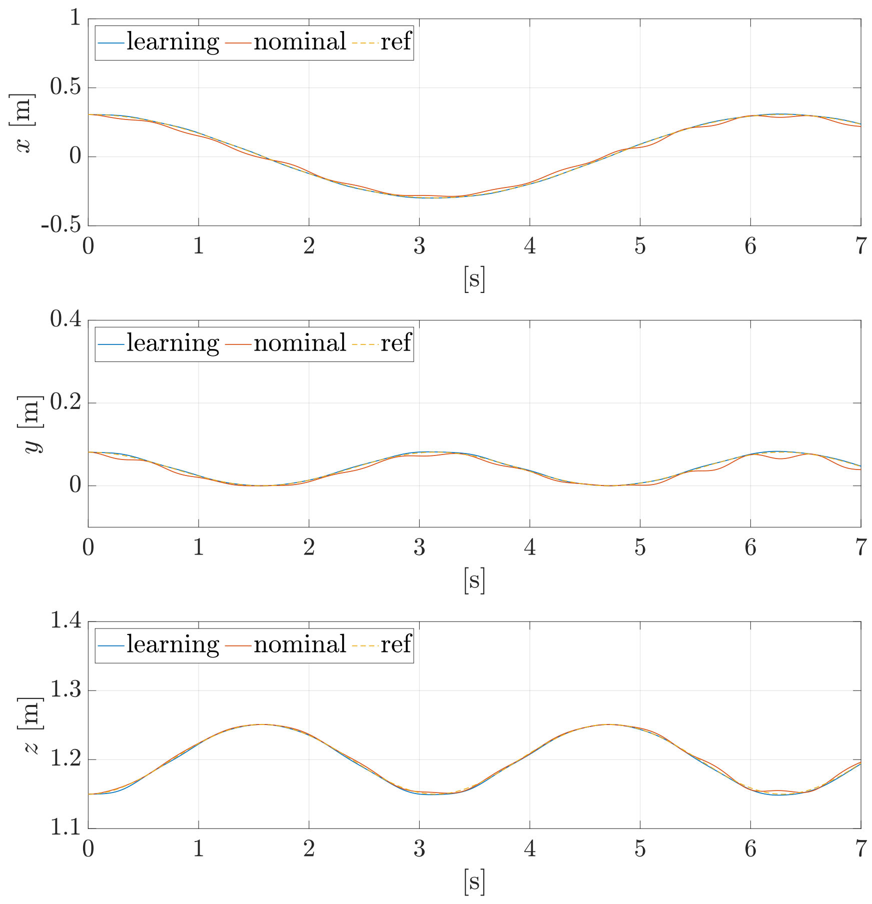

Figure 4.4. End-effector trajectory of the robot in the 3D cartesian space: in red with the nominal model, in blue with learning enabled.

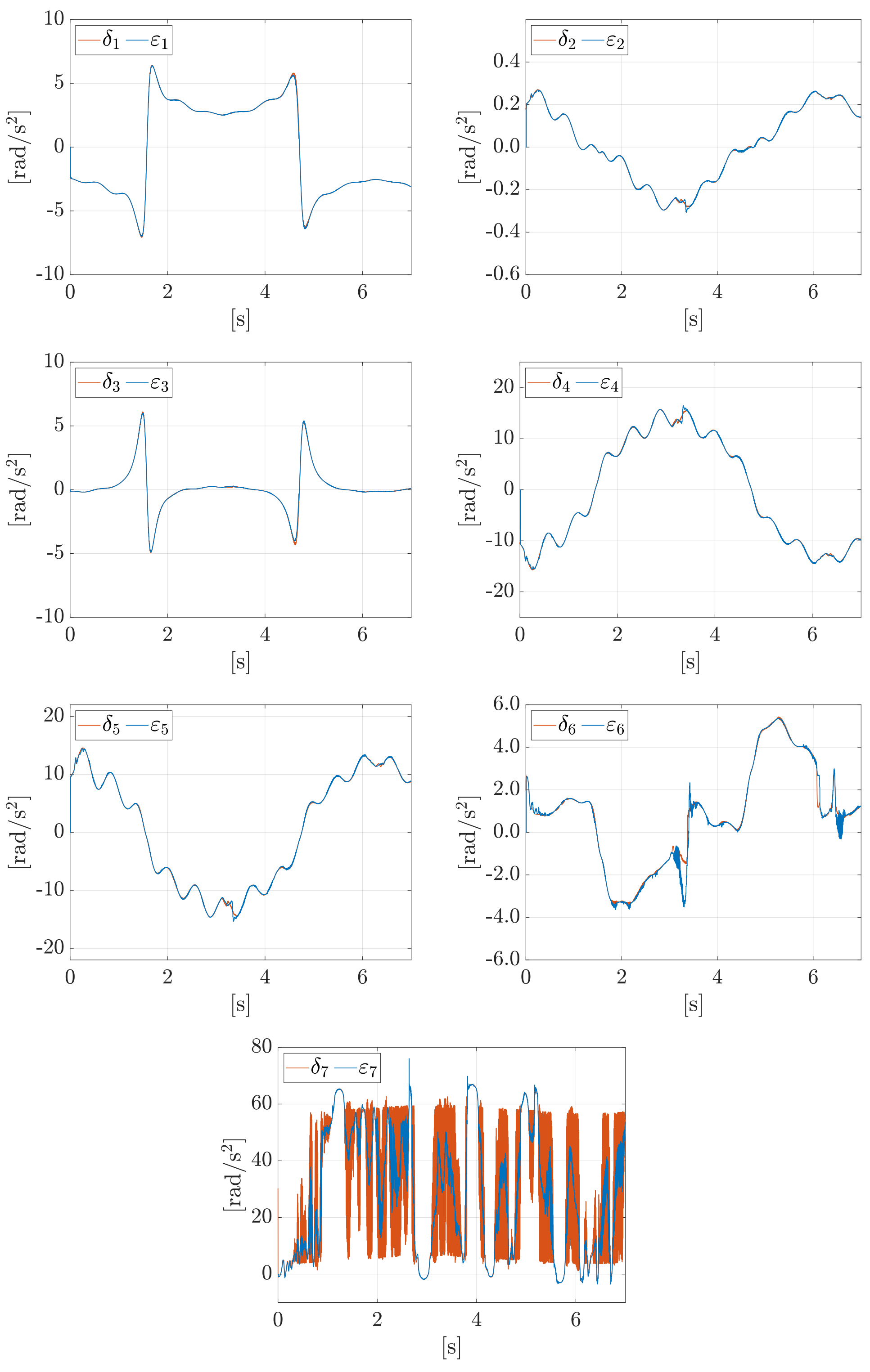

Figure 4.5. Comparison between the GP predictions $\boldsymbol{\varepsilon}$ and the mismatch signals $\boldsymbol{\delta}$ for each joint. The last joint, given its mass and inertia, is the more susceptible to the acceleration mismatch.

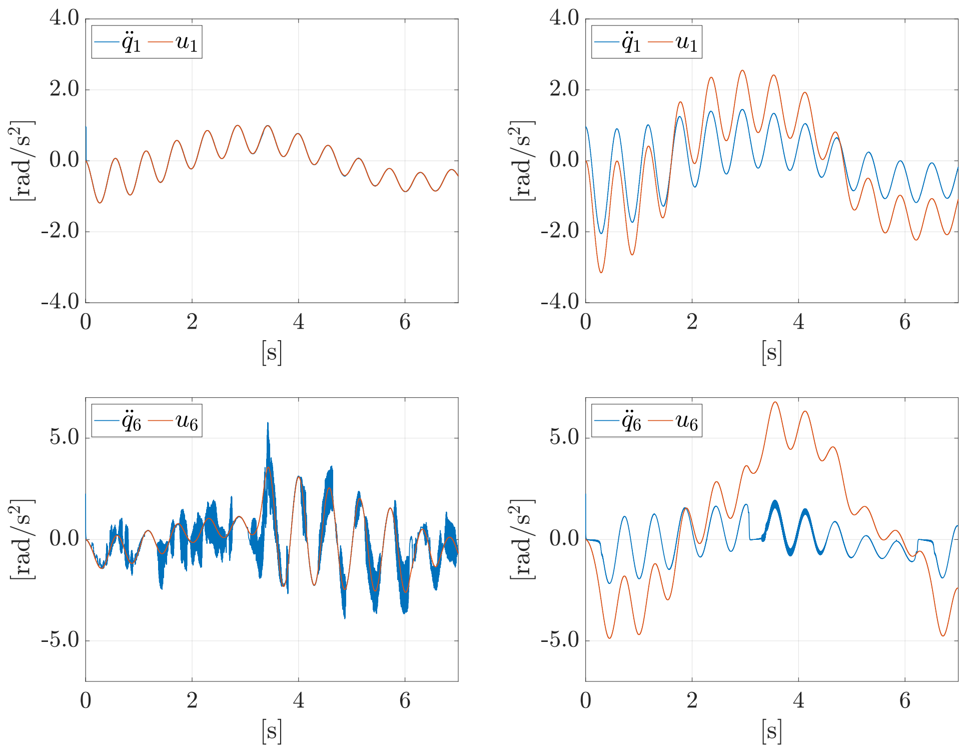

Figure 4.6. Comparison between the commanded accelerations $\boldsymbol{u}$ and the actual accelerations of the robot using our approach (left) and using a nominal controller (right) for joints 1 and 6.

Chapter 5

Online learning for planning and control of underactuated robots

5.1 Motivation and Contribution

Underactuation in mechanical systems occurs when there are less independent actuation inputs than generalized coordinates. This situation may be due to the nature of the mechanism, to its prevailing design, or it may be the result of an intentional choice aimed at reducing weight, cost or energy consumption. Many advanced robotic platforms are indeed underactuated, including manipulators with some passive joints, most underwater and aerial vehicles, legged robots, and nonprehensile manipulation systems.

An adverse effect of underactuation is that generic state space trajectories become unfeasible, since the dynamics of the passive degrees of freedom represents a set of second-order differential constraints that must be satisfied throughout any motion [99]; in practice, this limits the directions of instantaneous accelerations that can be commanded to the system. As a consequence, trajectory planning in the absence of obstacles, which is a relatively trivial issue for fully actuated robots, becomes a challenging problem in the presence of underactuation. Motion control is also made more difficult for these systems since full state feedback linearization cannot be achieved, and one has to deal directly with - actually, make use of - nonlinear, coupled dynamic effects.

In the literature, several model-based techniques have been proposed for planning and stabilizing motions of specific underactuated robots, with a notable emphasis on manipulators with passive joints [100]. In particular, the problem of state transfer between equilibria has been addressed mainly on two benchmark platforms, i.e., the Pendubot and the Acrobot; these are both 2R robots moving in the vertical plane with a single actuated joint (respectively, the first and the second). A classical approach is to use collocated or non-collocated Partial Feedback Linearization (PFL) in combination with energy-based controllers [101, 102]. Swing-up maneuvers of these robots have been achieved using passivity-based approaches [103, 104], orbit stabilization [105], impulse-momentum techniques [106], and sequential action control [107]. Typically, the maneuver includes a final balancing phase realized through an LQR (Sec. 3.1) designed around the target equilibrium.

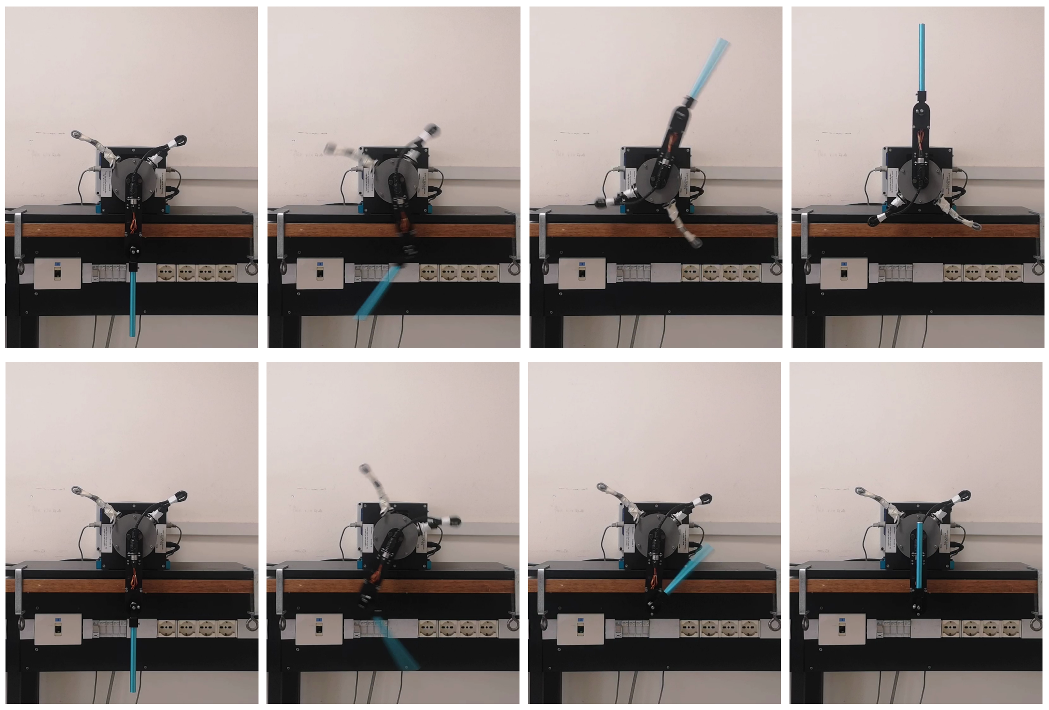
Figure 5.1. The Pendubot performing two different swing-up maneuvers using the proposed method: the targets are the up-up equilibrium (first row) and the down-up equilibrium (second row).

Although effective, the above approaches have two main limitations from the viewpoint of our work. First, the design techniques used for trajectory planning and control are invariably specific (or had to be specialized) for the considered robot, and sometimes also for the particular maneuver. Second, and even more important, all of them require an accurate knowledge of the robot dynamic model for successful performance. Exceptions are [108, 109], which propose robust control of the Pendubot via adaptive and fuzzy sliding modes respectively; however, these methods are tailored to the platform and do not tolerate large model uncertainties in practice.

To avoid the need for an accurate dynamic model, modern learning techniques have been applied for deriving feedforward and feedback control even for underactuated robots. In [110, 111] and [112], model-based RL procedures are proposed to generate robot control policies in a data-efficient way. However, this class of algorithms is not able in general to ensure the satisfaction of hard constraints in a specific robot task as explained in Sec. 2.2. An optimization-based iterative learning approach is used in [113, 114], where experience from previous robot trials is used to build incrementally the feedforward command needed to follow a desired output trajectory.

Other works that are more closely related to our approach have been published recently. In [115], a method for the swing-up of an Acrobot has been proposed which avoids the need for a model by using deep RL, requiring however a huge number of experiments for training. A robust control scheme for trajectory tracking under repetitive disturbances has been presented in [116] for a 3R planar manipulator with two actuators and one passive joint. The control design is tailored to this specific system, and cannot be easily extended to generic underactuated robots. In [117], a learning scheme is proposed to realize trajectory tracking of underactuated balance robots (e.g., a Furuta pendulum); because of the simpler balancing task, the reference trajectory of the active joints is not replanned and stabilization in the large is never addressed.

In this chapter, we build upon our learning method for fully actuated robots discussed in detail in Ch. 4, to devise an iterative approach for planning and controlling transfers between (stable or unstable) equilibria of underactuated robots in the presence of large dynamic uncertainties. The basic idea is to alternate off-line optimization-based planning and online PFL control, using regression to learn model corrections for the active and passive degrees of freedom. As a result, dynamically feasible state reference trajectories are learned and convergence to zero trajectory tracking error is obtained over the iterations. The main benefits of the proposed approach are:

- it applies to any underactuated robot;
- it applies to any state transfer maneuver;
- convergence is reached even in the presence of large uncertainties on the robot dynamics, requiring very few iterations in the considered case studies;
- more accuracy in the nominal dynamic model leads to even faster convergence;
- additional constraints (on state, on input, obstacle avoidance, etc.) can be explicitly taken into account in the optimization problem of the planning phase.

As an application, we provide an extensive evaluation of the performance of our approach on a Pendubot which must execute various swing-up maneuvers and state transfers between unstable equilibria (see Fig. 5.1). Furthermore, we provide simulation for a three links underactuated robots to show the generality of our approach.

This chapter is organized as follows. Section 5.2 introduces the dynamic model of underactuated robots, highlighting how model uncertainties affect the active and passive subsystems. The proposed iterative approach is presented in Sec. 5.3, discussing both the planning and the control phases and describing the data collection procedures with the regressors adopted for learning. In Sec. 5.4, we report on the application to the Pendubot, showing comparative simulation and experimental results. Finally, a summary of the proposed approach is presented and some general conclusions about the approach are drawn in Sec. 5.5.

5.2 Problem Formulation

For a robot with n degrees of freedom (dof) and $m < n$ actuators, the dynamics can be expressed [100] as

$$\boldsymbol{M}_{aa}(\boldsymbol{q})\ddot{\boldsymbol{q}}_a + \boldsymbol{M}_{ap}(\boldsymbol{q})\ddot{\boldsymbol{q}}_p + \boldsymbol{n}_a(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau} \tag{5.1}$$

$$\boldsymbol{M}_{pa}(\boldsymbol{q})\ddot{\boldsymbol{q}}_a + \boldsymbol{M}_{pp}(\boldsymbol{q})\ddot{\boldsymbol{q}}_p + \boldsymbol{n}_p(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{0}, \tag{5.2}$$

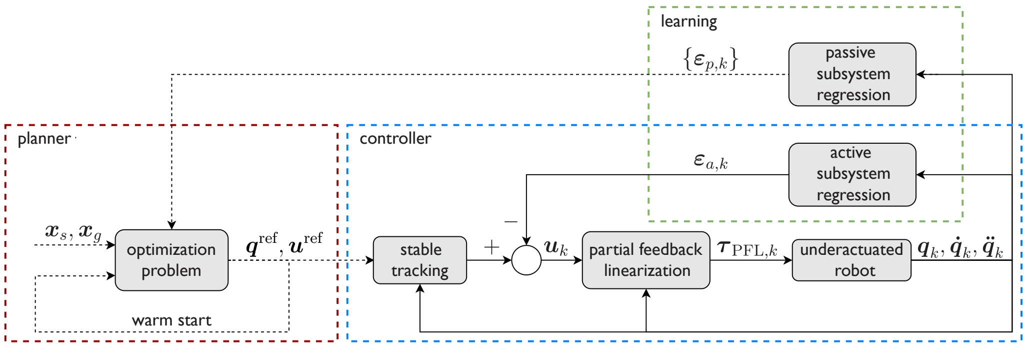

Figure 5.2. Block diagram of the generic iteration of the proposed algorithm. Solid signal lines represent data that are used at each time step, whereas dashed lines are data transferred at the end of the iteration.

where $\boldsymbol{q} = (\boldsymbol{q}_a, \boldsymbol{q}_p)$ is the n-dimensional configuration vector, with $\boldsymbol{q}_a$, $\boldsymbol{q}_p$ representing respectively the m active and the $n - m$ passive generalized coordinates. The inertia matrix $\boldsymbol{M}$ and the vector $\boldsymbol{n}$ of the remaining nonlinear terms are partitioned accordingly. The m generalized forces $\boldsymbol{\tau}$ only perform work on the $\boldsymbol{q}_a$ coordinates. We do not assume any structural control property (e.g., feedback linearizability or flatness) for system (5.1-5.2), nor any particular degree of underactuation.

In the presence of model perturbations (incorrect parameters and/or unmodeled dynamics), we are again in the same case of eqs. (4.4), where only the nominal terms $\hat{\boldsymbol{M}}$ and $\hat{\boldsymbol{n}}$ are known and available for control design.

Let $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$ be the robot state. Given a start and a goal equilibrium points, respectively denoted by $\boldsymbol{x}_s = (\boldsymbol{q}_s, \boldsymbol{0})$ and $\boldsymbol{x}_g = (\boldsymbol{q}_g, \boldsymbol{0})$, we want to plan and execute in a fixed time T a transfer motion from the start to the goal, while satisfying constraints on state and/or inputs, collectively expressed in the form $\boldsymbol{h}(\boldsymbol{q}, \boldsymbol{\tau}) \leq \boldsymbol{0}$. This *transfer between equilibria* problem is particularly challenging for robots that are not fully actuated because not all trajectories between two equilibria are feasible.

For the following developments, it is convenient to perform a preliminary nonlinear feedback aimed at exactly linearizing the *nominal* active dynamics. This collocated PFL controller is always well defined and takes the form

$$\boldsymbol{\tau}_{\mathrm{PFL}} = \left(\hat{\boldsymbol{M}}_{aa} - \hat{\boldsymbol{M}}_{ap}\hat{\boldsymbol{M}}_{pp}^{-1}\hat{\boldsymbol{M}}_{pa}\right)\boldsymbol{u} + \hat{\boldsymbol{n}}_a - \hat{\boldsymbol{M}}_{ap}\hat{\boldsymbol{M}}_{pp}^{-1}\hat{\boldsymbol{n}}_p, \tag{5.3}$$

where $\boldsymbol{u} \in \mathbb{R}^m$ is the new input, i.e., the acceleration of the active dofs. Using (5.3) and (4.4) in (5.1–5.2), we obtain the perturbed closed-loop dynamics

$$\ddot{\boldsymbol{q}}_a = \boldsymbol{u} + \boldsymbol{\delta}_a(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \tag{5.4}$$

$$\ddot{\boldsymbol{q}}_p = -\hat{\boldsymbol{M}}_{pp}^{-1}(\hat{\boldsymbol{n}}_p + \hat{\boldsymbol{M}}_{pa}\ddot{\boldsymbol{q}}_a) + \boldsymbol{\delta}_p(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}_a), \tag{5.5}$$

where $\boldsymbol{\delta}_a$ and $\boldsymbol{\delta}_p$ represent the cumulative effect of perturbations on the active and passive subsystems, respectively.

5.3 The Proposed Iterative Approach

The presence of model perturbations affects the considered planning and control problem at two levels. First, planning based on the nominal model would produce trajectories that may not be feasible, and in any case do not land at the goal equilibrium. Second, even when the reference trajectory is feasible, effective tracking is not achieved if the controller is designed on the nominal model.

In this section, we describe an iterative scheme for concurrent planning and control. At its core there is a *learning process* (Sects. 5.3.3–5.3.4) which continuously updates two regressors $\boldsymbol{\varepsilon}_a$ and $\boldsymbol{\varepsilon}_p$, respectively estimates of the perturbations $\boldsymbol{\delta}_a$ and $\boldsymbol{\delta}_p$ in (5.4–5.5). Both regressors are reconstructed from position measurements during robot motion.

Each iteration consists of an off-line *planning* phase and an online *control* phase. In the planning phase (Sec. 5.3.1), the nominal model is corrected by taking into account $\boldsymbol{\varepsilon}_p$; an optimization problem is then solved to compute a reference trajectory $\boldsymbol{q}^{\text{ref}}(t)$ leading this model to $\boldsymbol{x}_g$ at time T. In the control phase (Sec. 5.3.2), the robot tracks $\boldsymbol{q}^{\text{ref}}(t)$ under the action of a PFL control law given by (5.3), in which the corrective term $\boldsymbol{\varepsilon}_a$ is added to the commanded acceleration $\boldsymbol{u}$. During the motion, new data points are collected and used in the learning process. In this work, we employ Gaussian Processes regressors (Sec. 2.1.1) for reconstructing $\boldsymbol{\varepsilon}_a$ and $\boldsymbol{\varepsilon}_p$, given the good performance that this technique displays in the online learning context, with a squared exponential kernel (eq. 2.12). However, it is important to notice that other techniques such as Neural Networks, Generalized Linear Regression or Support Vector Machine could have been adopted without any modifications on the structure of the framework.

A block diagram of the generic iteration of the proposed approach is shown in Fig. 5.2.

5.3.1 Planning

In the planning phase, a reference trajectory is computed by solving a numerical optimal control problem for the underactuated robot. In particular, a prediction model is obtained by setting $\boldsymbol{\delta}_a = \boldsymbol{0}$ and $\boldsymbol{\delta}_p = \boldsymbol{\varepsilon}_p$ in eqs. (5.4–5.5):

$$\ddot{\boldsymbol{q}}_a = \boldsymbol{u} \tag{5.6}$$

$$\ddot{\boldsymbol{q}}_p = -\hat{\boldsymbol{M}}_{pp}^{-1}(\hat{\boldsymbol{n}}_p + \hat{\boldsymbol{M}}_{pa}\boldsymbol{u}) + \boldsymbol{\varepsilon}_p(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}). \tag{5.7}$$

In other words, we are assuming in (5.6) that partial feedback linearization has been achieved in spite of model perturbations. The rationale is that the control law will try to cancel $\boldsymbol{\delta}_a$ as much as possible using a correction term equal to its current estimate $\boldsymbol{\varepsilon}_a$ (see Sec. 5.3.2). Moreover, the available estimate $\boldsymbol{\varepsilon}_p$ of the perturbation on the passive subsystem has been used in (5.7). Upon convergence of the overall scheme, eq. (5.6) will become exact, and $\boldsymbol{\varepsilon}_p$ in (5.7) will eventually be equal to $\boldsymbol{\delta}_p$. In principle, we could have also included $\boldsymbol{\varepsilon}_a$ in the right-hand side of (5.6). The learning transient would be similar and, upon convergence, the obtained system behavior would be the same. However, the separate use of one regressor ($\boldsymbol{\varepsilon}_p$) in the planning phase and of the other ($\boldsymbol{\varepsilon}_a$) in the control phase proves to be computationally more efficient.

We consider a discrete-time setting in which the input $\boldsymbol{u}$ is piecewise-constant over N sampling intervals of duration $T_s = T/N$. Denoting by $\boldsymbol{f}(\cdot)$ a discretization of the state-space representation corresponding to (5.6–5.7), with the robot state

$\boldsymbol{x}_i = \boldsymbol{x}(t_i)$ and the starting and goal equilibrium points $\boldsymbol{x}_s$ and $\boldsymbol{x}_g$ defined in Sec. 5.2, the nonlinear optimization problem is written as

$$\min_{\boldsymbol{u}_0,\dots,\boldsymbol{u}_{N-1}} \sum_{i=0}^{N-1} J_s(\boldsymbol{x}_i, \boldsymbol{u}_i) + J_N(\boldsymbol{x}_N)$$

subject to

$$\begin{aligned} \boldsymbol{x}_{i+1} - \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) &= 0, \qquad i = 0\dots, N-1, \\ \boldsymbol{g}(\boldsymbol{x}_i) &\leq 0, \qquad i = 1, \dots, N, \\ \boldsymbol{h}(\boldsymbol{u}_i) &\leq 0, \qquad i = 0, \dots, N-1, \end{aligned}$$

with $\boldsymbol{x}_0 = \boldsymbol{x}_s$. The objective function is the sum of a stage cost J_s and a terminal cost J_N, both penalizing the state error with respect to the goal $\boldsymbol{x}_g$ and the control effort, while $\boldsymbol{g}$ and $\boldsymbol{h}$ represent state and input constraints, respectively. The cost terms take the form

$$J_s(\boldsymbol{x}_i, \boldsymbol{u}_i) = (\boldsymbol{x}_g - \boldsymbol{x}_i)^\top \boldsymbol{Q} (\boldsymbol{x}_g - \boldsymbol{x}_i) + \boldsymbol{u}_i^\top \boldsymbol{R} \boldsymbol{u}_i,$$

$$J_N(\boldsymbol{x}_N) = (\boldsymbol{x}_g - \boldsymbol{x}_N)^\top \boldsymbol{Q}_N (\boldsymbol{x}_g - \boldsymbol{x}_N)$$

where $\boldsymbol{Q}$, $\boldsymbol{Q}_N$ and $\boldsymbol{R}$ are positive-definite, symmetric matrices of weights. The solution of the NLP is a reference trajectory with the associated nominal input, represented by discrete sequences $\boldsymbol{q}^{\text{ref}} = \{\boldsymbol{q}_1^{\text{ref}}, \dots, \boldsymbol{q}_N^{\text{ref}}\}$ and $\boldsymbol{u}^{\text{ref}} = \{\boldsymbol{u}_0^{\text{ref}}, \dots, \boldsymbol{u}_{N-1}^{\text{ref}}\}$ respectively. The reference velocities $\dot{\boldsymbol{q}}^{\text{ref}} = \{\dot{\boldsymbol{q}}_1^{\text{ref}}, \dots, \dot{\boldsymbol{q}}_N^{\text{ref}}\}$ are also available.

To speed up convergence to a solution, one typically uses the reference trajectory of the previous iteration as a warm start when solving the current NLP. Furthermore, in our context this idea helps to diminish the number of iterations needed by our approach to convergence. In fact, each planned trajectory will be more similar to its previous one, being by construction local solutions of the previous optimization problems and probably near to the minimum of the successive NLP. This idea incentivizes learning a *local* model of the system instead of the global one, that is obviously more difficult to estimate correctly with little data. Notice that a similar behaviour can be obtained using the variance information of the GP (2.14) inside the NLP cost function. However, this can increase significantly the planning time, and furthermore it will not minimize directly the difference between two subsequent planned trajectories but only the distance with respect to the actual robot motions.

5.3.2 Control

In the control phase, the robot moves under the action of a digital control law, that for simplicity it is assumed with a control sampling interval T_s equal to the one in planning, aimed at driving $\boldsymbol{q}$ along the current reference trajectory $\boldsymbol{q}^{\text{ref}}$. To achieve stable tracking of $\boldsymbol{q}_a^{\text{ref}}$, the commanded acceleration $\boldsymbol{u}_k$ in $[t_k, t_{k+1})$ is chosen as

$$\boldsymbol{u}_k = \boldsymbol{u}_k^{\text{ref}} + \boldsymbol{K}_P(\boldsymbol{q}_{a,k}^{\text{ref}} - \boldsymbol{q}_{a,k}) + \boldsymbol{K}_D(\dot{\boldsymbol{q}}_{a,k}^{\text{ref}} - \dot{\boldsymbol{q}}_{a,k}) - \boldsymbol{\varepsilon}_{a,k}, \tag{5.8}$$

with $\boldsymbol{K}_P, \boldsymbol{K}_D > 0$. Here, the nominal input produced by the planner is used as feedforward term, and the current regressor $\boldsymbol{\varepsilon}_{a,k}$ has been added to cancel at best the perturbation $\boldsymbol{\delta}_a$ affecting the active subsystem (5.4). Note that as soon as $\boldsymbol{q}_a$ will be able to follow exactly $\boldsymbol{q}_a^{\text{ref}}$, the passive variables $\boldsymbol{q}_p$ will evolve as planned in the previous phase. Next, we use (5.3) to compute the generalized force as

$$\boldsymbol{\tau}_{\text{PFL},k} = \hat{\boldsymbol{B}}_k \boldsymbol{u}_k + \hat{\boldsymbol{\eta}}_k, \tag{5.9}$$

where

$$\hat{\boldsymbol{B}}_k = \hat{\boldsymbol{M}}_{aa}(\boldsymbol{q}_k) - \hat{\boldsymbol{M}}_{ap}(\boldsymbol{q}_k)\hat{\boldsymbol{M}}_{pp}^{-1}(\boldsymbol{q}_k)\hat{\boldsymbol{M}}_{pa}(\boldsymbol{q}_k)$$

and

$$\hat{\boldsymbol{\eta}}_k = \hat{\boldsymbol{n}}_a(\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k) - \hat{\boldsymbol{M}}_{ap}(\boldsymbol{q}_k)\hat{\boldsymbol{M}}_{pp}^{-1}(\boldsymbol{q}_k)\hat{\boldsymbol{n}}_p(\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k).$$

It should be noticed that different control laws can be used to derive the commanded acceleration $\boldsymbol{u}_k$. For example, in our short paper [118] we have employed a Nonlinear MPC given the remaining nonlinearities of the unactuated part of the system. Alternatively, one can use without any major difference a discrete linear time-varying LQR (Sec. 3.1), linearizing completely the partial feedback linearized system on the planned trajectory.

5.3.3 Dataset collection procedure for the active dofs

The collection procedure of the datapoint needed to learn $\boldsymbol{\varepsilon}_{a,k}$ follow closely the description carried out in the previous chapter in Sec. 4.3.1. In fact, from eq. (5.4) we may write

$$\boldsymbol{\delta}_{a,k} = \ddot{\boldsymbol{q}}_{a,k} - \boldsymbol{u}_k. \tag{5.10}$$

and in view of eq. (5.10), a new data point is generated at the k-th control step as

$$\boldsymbol{X}_{a,k} = (\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \boldsymbol{u}_k) \qquad \boldsymbol{Y}_{a,k} = \ddot{\boldsymbol{q}}_{a,k} - \boldsymbol{u}_k.$$

with the acceleration $\ddot{\boldsymbol{q}}_{a,k}$ to be reconstructed in this work numerically. We note that the actual acceleration is functionally dependent through (5.4) on the robot state $(\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k)$ and on the commanded acceleration $\boldsymbol{u}_k$, i.e., on the input $\boldsymbol{X}_{a,k}$ of the regression scheme.

Every time a new data point is available, it is immediately used to update the regressor $\boldsymbol{\varepsilon}_a$, maintaining a queue of dimension d that mixes the most recent data-points along with the most informative chosen according to the information gain criterion [27]. However, the hyperparameters of the kernel function (2.12) are only updated at the end of each iteration due to the time complexity needed to perform a tuning procedure.

5.3.4 Dataset collection procedure for the passive dofs

To learn an estimate $\boldsymbol{\varepsilon}_p$ of the model perturbation $\boldsymbol{\delta}_p$, we compare the commanded and the actual acceleration for the passive dofs. In fact, from eq. (5.5) we have

$$\boldsymbol{\delta}_{p,k} = \ddot{\boldsymbol{q}}_{p,k} + \hat{\boldsymbol{M}}_{pp,k}^{-1}(\hat{\boldsymbol{n}}_{p,k} + \hat{\boldsymbol{M}}_{pa,k}\,\ddot{\boldsymbol{q}}_{a,k}). \tag{5.11}$$

Given numerical approximations of the actual accelerations $\ddot{\boldsymbol{q}}_{a,k}$ and $\ddot{\boldsymbol{q}}_{p,k}$, a new data point is generated at the k-th step as

$$\begin{aligned}\boldsymbol{X}_{p,k} &= (\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \ddot{\boldsymbol{q}}_{a,k}) \\ \boldsymbol{Y}_{p,k} &= \ddot{\boldsymbol{q}}_{p,k} + \hat{\boldsymbol{M}}_{pp,k}^{-1}(\hat{\boldsymbol{n}}_{p,k} + \hat{\boldsymbol{M}}_{pa,k}\,\ddot{\boldsymbol{q}}_{a,k}).\end{aligned}$$

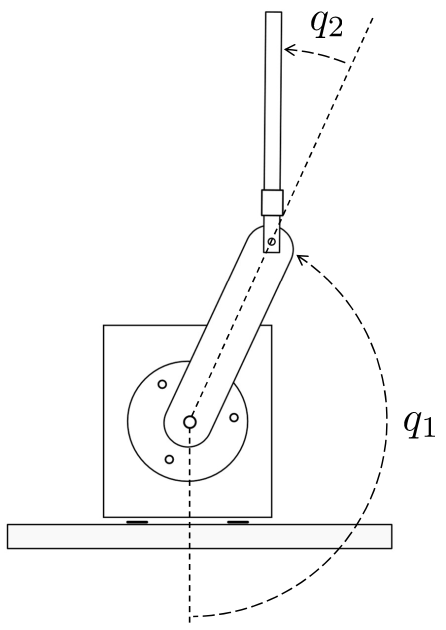

Figure 5.3. The Pendubot and its generalized coordinates.

Differently from $\boldsymbol{\varepsilon}_a$, *all* the data points computed during the iteration are used to update the passive subsystem regressor $\boldsymbol{\varepsilon}_p$ at the end of each trial; in fact, since the planning phase is performed off-line, the complexity associated to an exact regression does not represent a problem here. As before, the hyperparameters of the kernel function (2.12) are also updated at the end of the iteration.

5.4 Results

The proposed approach has been validated through simulations and experiments on the Pendubot, a two-link arm moving in the vertical plane with an active joint at the shoulder and a passive joint at the elbow ($\boldsymbol{q}_a = q_1$ and $\boldsymbol{q}_p = q_2$). See Fig. 5.3 for the definition of the generalized coordinates and the Appendix A.1 for the dynamic model and the nominal parameter values for our Pendubot prototype.

In the following, we will address the problem of executing various transfer motions between equilibria in the presence of severe uncertainty on the dynamic model. The proposed iterative method is used to steer the Pendubot to the basin of attraction of an LQR balancing controller designed around the goal state. The latter is obviously needed to stabilize the robot after the planning horizon T.

The discretized state-space model used in the planning phase has been obtained by Euler method. The sampling interval is set to $T_s = 10$ ms. The cost function J in the NLP includes two quadratic terms that penalize the state error with respect to the goal $\boldsymbol{x}_g = (q_{1,g}, q_{2,g}, 0, 0)$ as well as the control effort, while the optimization is performed in MATLAB using the `fmincon` function, which implements a Sequential Quadratic Programming method (Sec. 3.2). The joint velocities are bounded as $|\dot{q}_1| \leq 8$ rad/s and $|\dot{q}_2| \leq 15$ rad/s. Finally, terminal constraints are included to guarantee convergence at time T to the following basin of attraction of the balancing

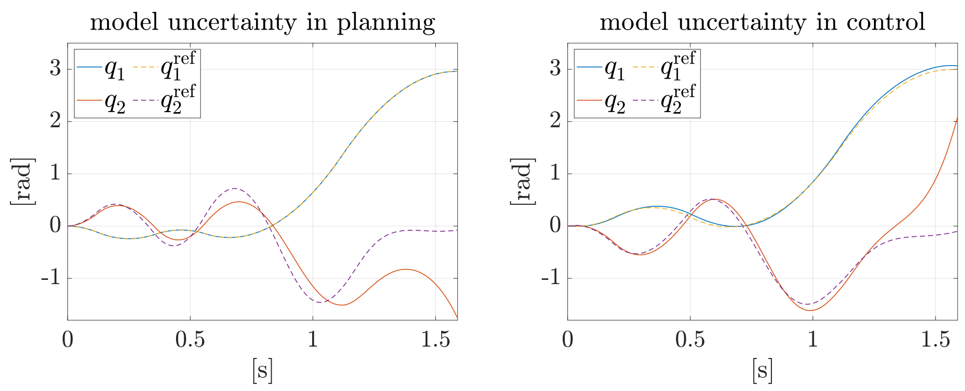

Figure 5.4. Simulation scenario 1 (*swing-up to* $\boldsymbol{q}^{\text{u-u}}$): results without learning. Left: Using the nominal model for planning and the true model for control. Right: Vice versa.

controller

$$|q_{j,N} - q_{j,g}| \leq 0.2, \qquad |\dot{q}_{j,N}| \leq 0.5, \qquad j = 1, 2.$$

We were able to obtain convergence to the basis of attraction for different values of the weights $\boldsymbol{Q}$, $\boldsymbol{Q}_N$ and R in the cost function. Still, they play an important role in shaping the final trajectory of the robot, e.g. lowering the joints velocities, the required control input etc.

In the control phase, the PD gains in (5.8) are chosen as $K_P = 50$ and $K_D = 20$, while the sampling interval is again 10 ms. While all data points (with n_d equal to N times the number of iterations so far) are considered for updating $\boldsymbol{\varepsilon}_p$, the maximum number of data points used for computing $\boldsymbol{\varepsilon}_a$ in real-time is $d = 180$. With this choice, the GP prediction takes around 7 ms on our PC equipped with an Intel i7-4770@3.40 GHz processor. It should be noticed that for a higher dofs system with more than one actuated joint, the prediction time can remain equal if multiple GP are learned and their predictions are shared across the cores of the processor. Given our computer, a similar computational complexity can be observer up to a maximum of 4 actuated joints.

The readers can refer to the following link `https://www.youtu.be/1aKG_8gfvk` for clips of all the simulations and experiments shown in the following.

5.4.1 Simulation Results

Two scenarios of transfer between equilibrium states will be presented. To show that the proposed method can achieve robust performance in the presence of severe model perturbations, we perturbed for control design the nominal values of the Pendubot parameters (see Table A.1), increasing by 30% the link masses m_1 and m_2 and reducing by the same percentage the distances d_1 and d_2 of the centers of mass of the two links from their respective joints. The barycentral inertia I_1 and I_2 were changed accordingly.

In the first scenario, the start configuration is $\boldsymbol{q}_s = (0, 0)$ while the goal is the *up-up* configuration $\boldsymbol{q}_g = \boldsymbol{q}^{\text{u-u}} = (\pi, 0)$, corresponding to a transfer from a stable to an unstable equilibrium (*swing-up*). The planning horizon is chosen as $T = 1.6$ s ($N = 160$).

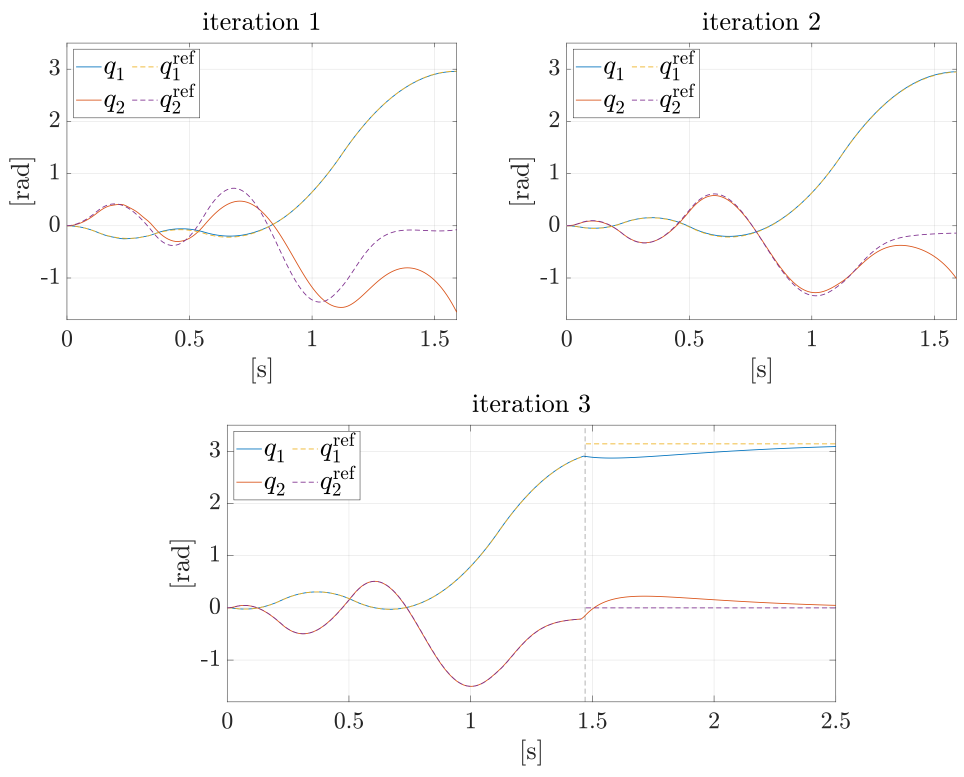

Figure 5.5. Simulation scenario 1 (*swing-up to* $\boldsymbol{q}^{\text{u-u}}$): results with the proposed approach. Just before the end of the third iteration, the state has converged to the basin of attraction of the balancing controller, which is then activated (as indicated by the vertical dashed line).

To highlight the necessity of learning in both the planning and control phases, we have preliminarily considered two complementary situations where learning is not used. Figure 5.4, left, refers to the first situation, in which we use the nominal model for planning and the true model for control. The result shows that planning the motion of an underactuated robot based on an inaccurate model produces dynamically unfeasible trajectories, that cannot be tracked in spite of the ideality of the controller. Vice versa, in Fig. 5.4, right, the true model is used for planning and the nominal for control. As expected, the inaccuracy of the controller prevents the completion of the swing-up maneuver.

Next, we tested the proposed approach on the same scenario, obtaining the results in Fig. 5.5. After three iterations, the Pendubot is able to track with sufficient accuracy the planned trajectory, ultimately entering the basin of attraction of the balancing controller to complete the swing-up maneuver. This shows that, in spite of the very large model uncertainty, the learning component of our method is able to reconstruct the correct model in a few iterations. Further iterations of the planning-control sequence do not change significantly the resulting motion.

To put our result in perspective, we have applied to this scenario also the passivity-based swing-up method proposed in [104], using the same balancing controller in the final phase. As shown in Fig. 5.6, the method works perfectly if the robot model is exactly known, but is unable to complete the maneuver in the presence of the

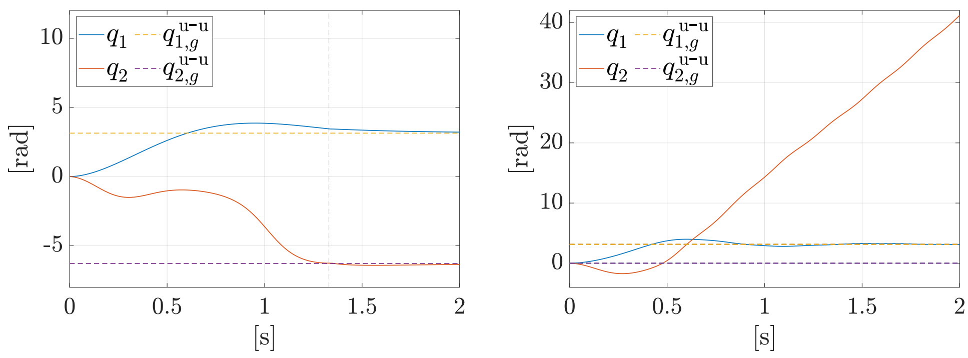

Figure 5.6. Simulation scenario 1 (*swing-up to* $\boldsymbol{q}^{\text{u-u}}$): results with the method in [104]. Left: assuming exact model knowledge the state enters the basin of attraction of the LQR controller at $t = 1.3$ s circa. Right: with the same model uncertainty of Fig. 5.5, convergence is not achieved.

.

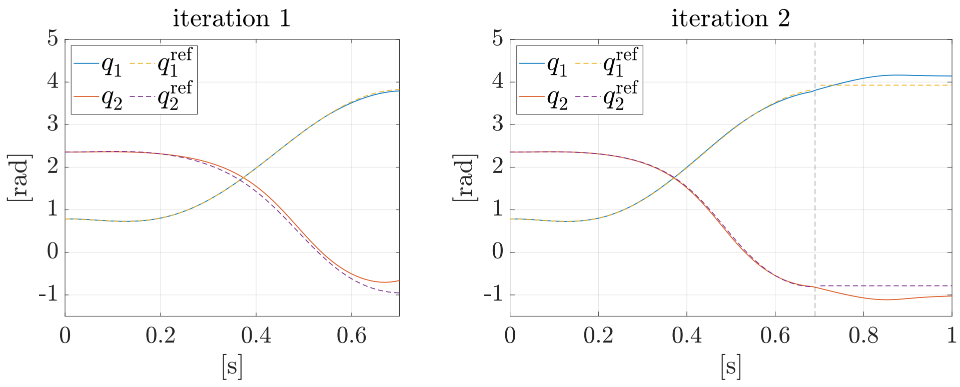

Figure 5.7. Simulation scenario 2 (*transfer between unstable equilibria*): results with the proposed approach. Two iterations are needed to achieve convergence.

considered model uncertainty. In particular, while the first joint still converges to its target, the passive joint drifts away very quickly.

In the second scenario, the start is $\boldsymbol{q}_s = (\pi/4, 3\pi/4)$ while the goal is $\boldsymbol{q}_g = (5\pi/4, -\pi/4)$; this amounts to a transfer between two unstable equilibria. The planning horizon is chosen as $T = 0.7$ s ($N = 70$). The results are shown in Fig. 5.7. Two iterations are now sufficient to reach the basin of attraction of the balancing controller, thus completing the maneuver correctly. Indeed, a closer look at the joint motion reveals that in both iterations the transfer is performed with the second link approximately vertical, a situation which inherently reduces the effect of the uncertain dynamic parameters, leading to a faster convergence.

We have performed further simulations on a 3R Pendubot (Fig. 5.8) with *two* passive joints that must execute a swing-up maneuver to $\boldsymbol{q}^{\text{u-u-u}}$ under perturbed conditions similar to scenario 1. Once again convergence was achieved in 3 iterations, a result suggesting that our method performs effectively even for higher degrees of underactuation.

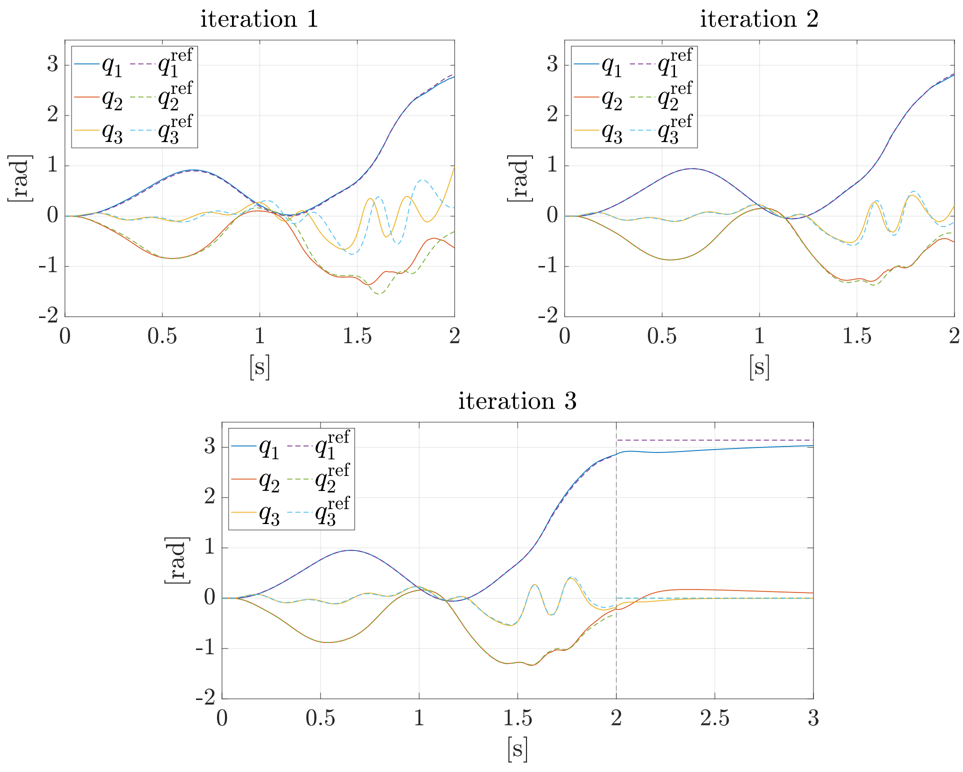

Figure 5.8. Simulation scenario 3 (*swing-up to* $\boldsymbol{q}^{\text{u-u-u}}$): results with the proposed approach. Three iterations are needed to achieve convergence.

5.4.2 Experimental Results

The proposed method has also been tested experimentally on our Pendubot prototype, using again the nominal model in A.1 for planning and control design. Joint velocities and accelerations are obtained in real-time via filtered numerical differentiation of encoder measurements. In our prototype, the encoders have a resolution of 1/8192 for the first joint while 1/4096 for the second. To further remove the noise affecting the learning dataset for the passive dofs, we used a non-causal Savitzky-Golay filter to compute $\ddot{q}_2$. It works trying to interpolate a low-degree polynomial on a moving horizon window, a procedure that is known as convolution. Since the learning of the passive joint is performed offline, there is no problem related to its speed or to the non-causality.

The first experiment replicates the swing-up scenario to $\boldsymbol{q}^{\text{u-u}}$ of Sec. 5.4.1, using the same planning horizon of $T = 1.6$ s ($N = 160$). The results are shown in Fig. 5.9. Only two iterations are required for our method to enter the basin of attraction of the LQR controller. The combination of online learning of the active joint dynamics together with the off-line re-planning of both joint trajectories, driven by the regressor built for the passive joint dynamics, allows a successful execution of the swing-up maneuver. When comparing the tracking errors between the single run without learning (Fig. 5.9, first image) and the first iteration of the method (Fig. 5.9, second image), no major changes are observed for the active joint q_1, whereas the passive joint q_2 behaves quite differently toward the end of the motion. In both cases, the error diverges (the second link falls in two opposite directions) because

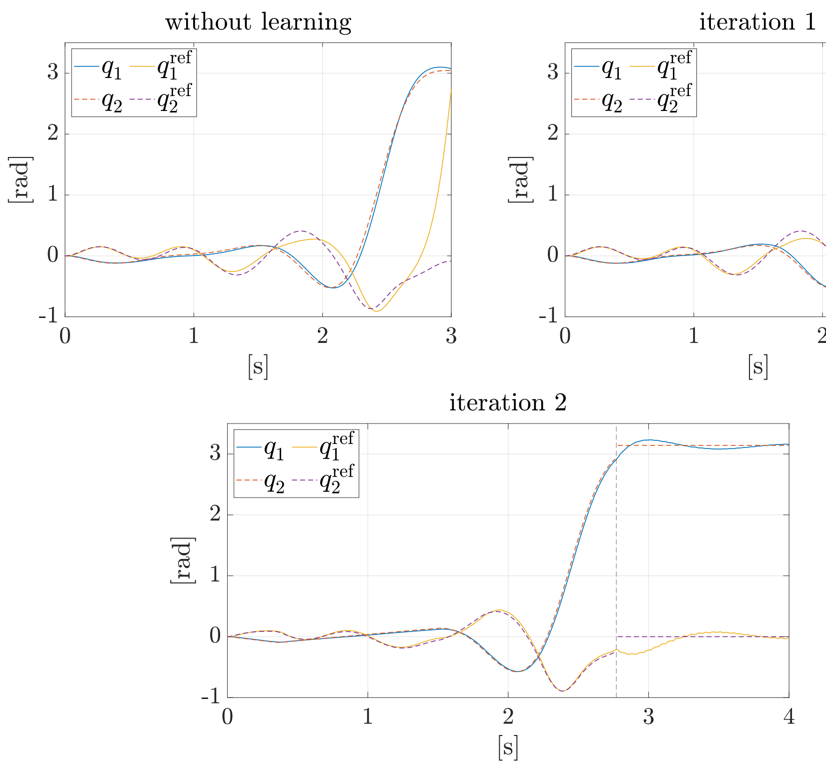

Figure 5.9. Experimental scenario 1 (*swing-up to* $\boldsymbol{q}^{\text{u-u}}$): results with the proposed approach. For comparison, the top-left image shows the results without learning, i.e., when partial feedback linearization and stable tracking for the first joint are computed on the nominal model.

the currently planned trajectory is still dynamically unfeasible for the Pendubot. Already at the second iteration (third image), the robot is correctly driven to the basin of attraction of the desired equilibrium (the LQR stabilizer is triggered at about $t = 1.4$ s). Performing a third planning-control iteration shows no significant variation of the obtained reference trajectory and control accuracy.

Table 5.1 offers further insight on the performance in both scenarios. In particular, it shows that the tracking accuracy for q_1 does not change significantly over the iterations, while the evolution of q_2 gets increasingly closer to the planned trajectory, as the latter approaches feasibility thanks to the model learning procedure.

Table 5.1. Tracking root mean squared error [rad] in the experiments

	scenario 1		scenario 2	
	q_1	q_2	q_1	q_2
without learning	0.045	0.191	0.056	0.320
iteration 1	0.035	0.623	0.022	0.470
iteration 2	0.037	0.038	0.023	0.034
iteration 3	0.036	0.036	–	–

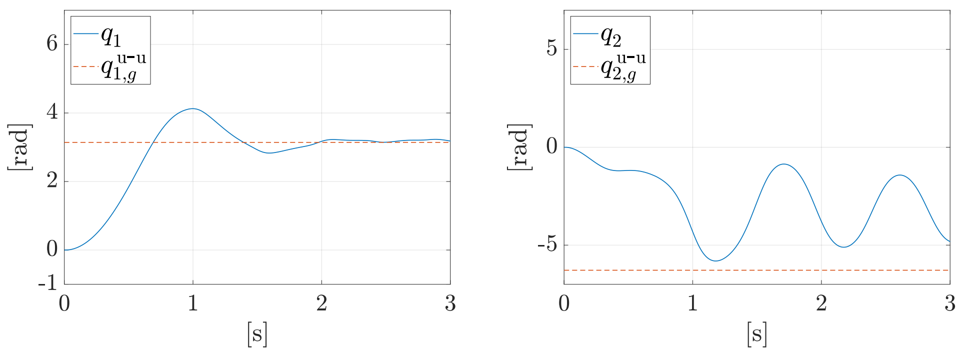

Figure 5.10. Experimental scenario 1 (*swing-up to* $\boldsymbol{q}^{\text{u-u}}$): results with the method in [104].

For comparison, Fig. 5.10 shows the experimental results obtained in this scenario with the method of [104] under the same nominal information on the robot dynamic model. While the active joint converges to its desired goal, the second joint oscillates (with a light damping due to friction) without ever entering the basin of attraction of the stabilizing controller. Therefore, we can claim that the learning procedure makes the proposed method able to withstand a level of model uncertainty that is not tolerated by purely model-based controllers.

The second experiment is again a swing-up scenario, but the goal is now the *down-up* configuration $\boldsymbol{q}^{\text{d-u}} = (0, \pi)$. The planning horizon has been set to $T = 2$ s ($N = 200$). As shown in Fig. 5.11, also in this case the learning procedure allows to complete the maneuver successfully after two iterations (see also the second column in Table 5.1).

5.5 Chapter Summary

In this chapter we have proposed an iterative method for planning and controlling motions of underactuated robots in the presence of model uncertainty, built on our previous work described in Ch. 4. The method hinges upon a learning process which estimates the induced perturbations on the dynamics of the active and passive dofs. Each iteration includes an off-line planning phase and an online planning phase, which take advantage of the learned data to improve the feasibility of the planned trajectory and the accuracy of its tracking.

The proposed approach was validated by application to the Pendubot, a well-known underactuated platform consisting of a 2R planar robot with a passive elbow joint. In particular, numerical simulations of our iterative method starting with considerable errors in the nominal dynamic parameters ($\pm 30\%$ of the true values) have shown that swing-up maneuvers and transfers between unstable equilibria can be executed successfully after very few iterations. This remarkable performance was confirmed in experimental tests on a real Pendubot. An additional simulation was performed on a more complex 3R planar manipulator, showing that our approach generalizes even in the case of higher dofs.

In addition to the applicability to general underactuated systems and independence from the specific maneuver, a further aspect of our method that deserves to be emphasized is that no torque measurement is required. In fact, only positions,

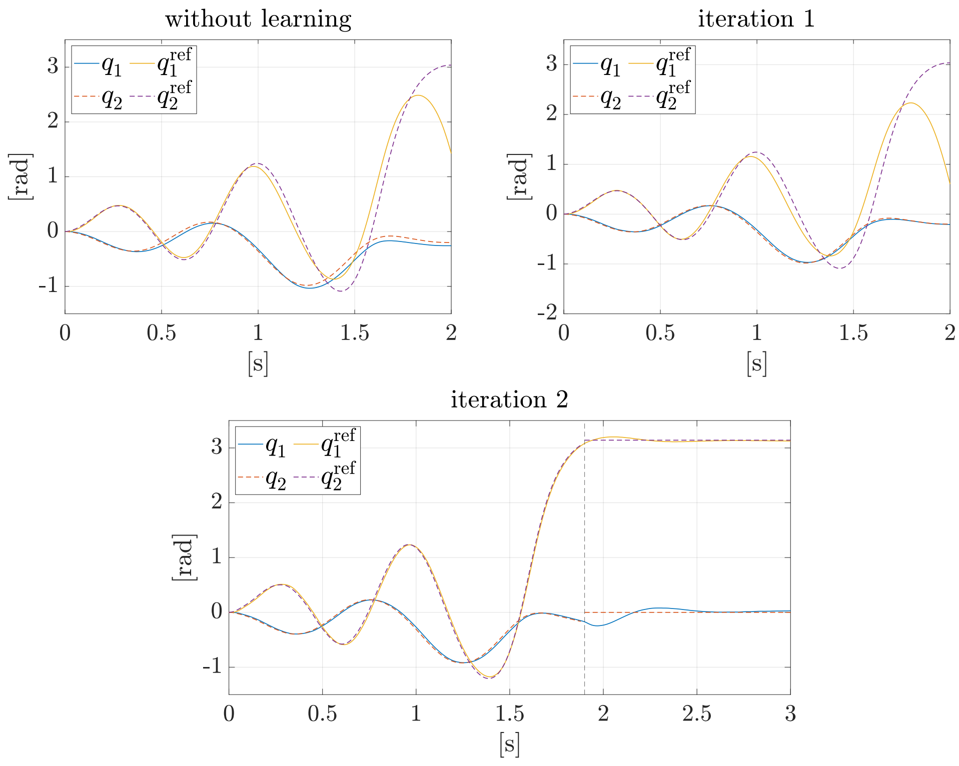

Figure 5.11. Experimental scenario 2 (*swing-up to* $\boldsymbol{q}^{\text{d-u}}$): results with the proposed approach. Just before the end of the second iteration the state converges to a region where the LQR controller can be successfully activated.

velocities and accelerations must be available, so that implementation is possible using only encoders. Another interesting feature is the possibility to incorporate constraints on the robot states and/or inputs in the planning phase, as well as to handle (without any modification) also the presence of repetitive external disturbances.

With this chapter we conclude the description of our works in the field of model learning.

Chapter 6

Enforcing constraints over learned policies via Nonlinear MPC

6.1 Motivation and Contribution

Reinforcement Learning, discussed previously in Sec. 2.2, in recent years has gained a lot of popularity in the robotics community thanks to its promise to deliver optimal control laws learned just from data and that do not require any specialized process of control synthesis. Nonetheless, real robotics applications pose many challenges which, still now, are not yet completely solved by the research community, such as the presence of continuous state spaces and control, the need for data efficiency (i.e. minimum number of interactions with the environment) if the learning procedure is performed directly on the robot, or in alternative the need of accurate of physics simulator if the learning is carried out offline, and finally the necessity of imposing state and control constraints on the final control law which can guarantee, for example, safety conditions.

In the last two decades many efforts have been devoted to improving the effectiveness of RL in robotics scenarios. Policy Search methods (Sec. 2.2.2) represent a first attempt to overcome the issue of directly estimating the optimal cost-to-go for each state, a problem that becomes rapidly intractable for robotics applications given the dimension of the state and the input space. From PS, more recently other solutions have been proposed to deal with continuous control-state space such as Deep Deterministic Policy Gradient (DDPG) [62] and Proximal Policy Optimization [119]. Moreover, in the literature it has been shown that PS approaches are effectively more data-efficient with respect to value function approximation methods (Sec. 2.2.1), and if a model of the controlled system is learned in combination with a control policy, such as in [110], just few iterations on the real robots are usually required to obtain a control policy which furthermore reduces to a minimum the need of learning on a simulator.

Fewer works instead consider the problem of constraints satisfaction. In RL this issue is usually addressed by extending the reward with a penalty term. This approach has however several drawbacks and feasibility is not always assured. [120]

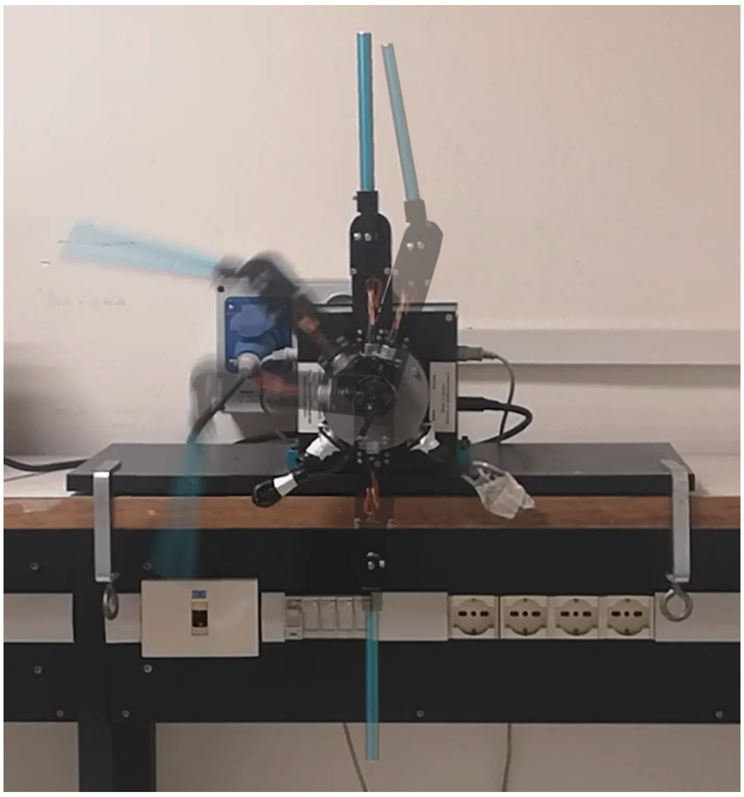

Figure 6.1. The Pendubot performing a swing-up maneuver with the proposed approach.

proposes an actor-critic approach based on a constrained Markov decision process formulation, which is solved by approximations. Similarly, in [121] a penalization term is added to limit constraint violations, thus increasing the safety of the learned policy. Another recent work [122] introduces an algorithm to learn the viability kernel directly in the control-state space. Even though it provides an estimation of a safety measure while learning, this procedure is conservative by construction and might converge slowly in a large control-state space. On the other hand, from the control community, different tools exist to deal explicitly with state and input constraints, such as Model Predictive Control (Sec. 3.2). The possibility of including input and state constraints in the optimization problem, together with its inherent robustness, makes MPC one of the most successful strategies. Even though a large number of algorithms have been developed for efficiently computing a solution, in general, for nonlinear control problems, the use of MPC in real time is typically not straightforward. Usually a simplified prediction model is convenient and, when the final state is not reachable in a small time frame, the knowledge of a reference trajectory for the entire task is required.

Many works aim at achieving better performance through the combination of MPC with data driven approaches. One of the first attempt to combine RL and MPC has been presented by [123], where an offline estimation of the value function is used as a final cost in an unconstrained iterative linear quadratic regulator. [124] propose iterative learning of a term in the objective function to incorporate long-term reasoning into the MPC. Other works have applied data driven techniques to learn a predictive model for MPC [125, 126]. More recent contributions towards the combination of MPC and RL are due to [127, 128, 129, 130].

Here we introduce an algorithm for combining MPC and RL that tries to overcome the aforementioned issues. We propose an optimal controller based on state-of-the-art Nonlinear MPC solvers that allow for constraints satisfaction and can be used in real-time on challenging robotic systems. We explore how the continuous interaction between the control policy and the MPC can take care of constraints satisfaction in a real application scenario. The performance of our method is tested both in simulation and experiments on a challenging underactuated robotic system, the same used in Ch. 5, the Pendubot (see Fig. 6.1).

This chapter is organized as follows. Sec. 6.2 describes the proposed approach going into the details of the control architecture, the RL algorithm and the NMPC formulation. Simulation and experimental results on the Pendubot are presented in Sec. 6.3. Finally, future developments are briefly discussed in Sec. 6.4.

6.2 The Proposed Approach

The proposed method consists of two different phases. In the first phase any RL approach can be used to learn an offline control policy that solves the desired task. In our case we use Deep Deterministic Policy Gradient (DDPG) from [62]. In the second phase, once a control policy is learned, we can employ this information online to provide guidance for an MPC. Here we define an NMPC problem and provide an online solution using the Real-Time Iteration scheme (Sec. 3.3), which is an SQP variant. At each time step, starting from the current state, the learned policy is applied to a simulated robot model over a fixed prediction horizon. The resulting control-state trajectory is passed as a reference to the cost function of the NMPC, which allows to transfer the policy learned offline into an optimal control problem to enforce the constraints satisfaction. Afterwards, the resulting optimal control action is applied to the physical robot, which in turn reaches a new state where the procedure is then restarted. A block scheme of the proposed algorithm is shown in Fig. 6.2.

Our method is capable to find a successful solution under the assumption that the control policy has learned a sequence of actions to accomplish the given task at least in a subset of the feasible region. In practice, an extensive training phase and a large exploration of the control-state space can ensure the validity of the aforementioned hypothesis.

6.2.1 Offline Policy Learning

Let us denote with $\boldsymbol{\pi}(\boldsymbol{x})$ the control policy, and with $\boldsymbol{u}_k$, $\boldsymbol{x}_k$ respectively the control action and the system state defined at time k. DDPG is an actor-critic algorithm where the control policy $\boldsymbol{\pi}(\boldsymbol{x}|\boldsymbol{\theta}^{\boldsymbol{\pi}})$ and the Q-value function $\boldsymbol{Q}(\boldsymbol{x}, \boldsymbol{\pi}(\boldsymbol{x}|\boldsymbol{\theta}^{\boldsymbol{Q}}))$ are both approximated with two neural networks, where $\boldsymbol{\theta}^{\boldsymbol{\pi}}$ and $\boldsymbol{\theta}^{\boldsymbol{Q}}$ are their parameters. The actor $\boldsymbol{\pi}$, given the actual state $\boldsymbol{x}_k$, outputs a control action $\boldsymbol{u}_k$, while the critic $\boldsymbol{Q}$ provides an estimate of the Q-value function for the pair $(\boldsymbol{u}_k, \boldsymbol{x}_k)$. During the offline training phase, at each time step, the agent applies an action $\boldsymbol{u}_t = \boldsymbol{\pi}(\boldsymbol{x}_k)$ to the simulated robot and collects a reward r_k from the environment. The generated

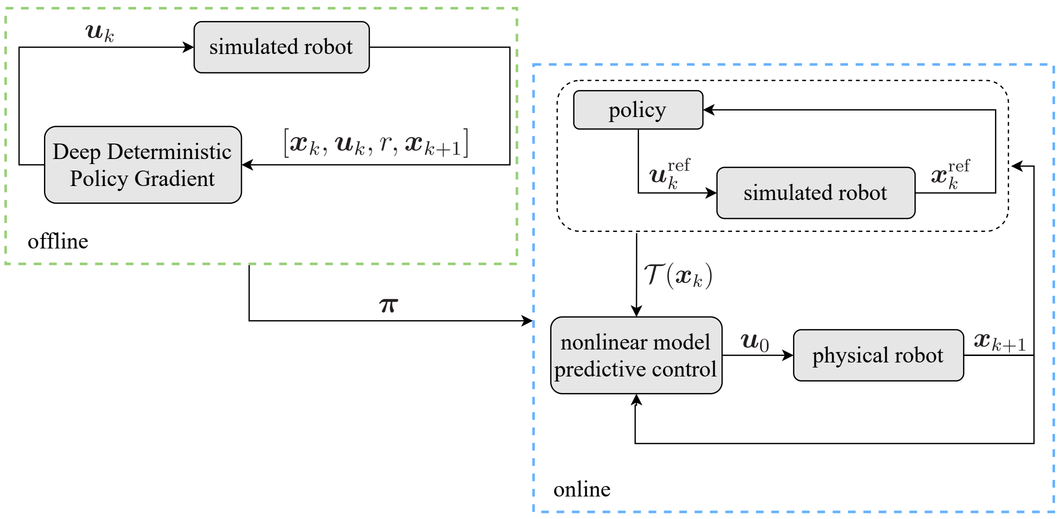
$\boldsymbol{u}_k$
simulated robot
Deep Deterministic Policy Gradient
$[\boldsymbol{x}_k, \boldsymbol{u}_k, r, \boldsymbol{x}_{k+1}]$
offline
$\boldsymbol{\pi}$
policy
$\boldsymbol{u}_k^{\text{ref}}$
simulated robot
$\boldsymbol{x}_k^{\text{ref}}$
$\mathcal{T}(\boldsymbol{x}_k)$
nonlinear model predictive control
$\boldsymbol{u}_0$
physical robot
$\boldsymbol{x}_{k+1}$
online

Figure 6.2. Block scheme of the proposed approach. At each control step, the policy $\boldsymbol{\pi}$ is queried to obtain an up-to-date trajectory.

experience represented by $(\boldsymbol{x}_k, \boldsymbol{u}_k, r_k, \boldsymbol{x}_{k+1})$ is stacked inside the Replay Buffer (RB), an array that contains a fixed number of elements I. To stabilize the learning procedure, two additional copies of the actor, $\boldsymbol{\pi}'(\boldsymbol{x}|\boldsymbol{\theta}^{\pi'})$ and the critic $\boldsymbol{Q}'(\boldsymbol{x}, \boldsymbol{\pi}(\boldsymbol{x})|\boldsymbol{\theta}^{\boldsymbol{Q}'})$ are used in the algorithm in order to avoid abrupt changes in their value due to the optimization routine. In fact, during training their parameters are usually transferred to the main networks with a weighting factor α_t that can be freely tuned.

DDPG alternates between updating the Q-value function and the control policy. The learning process is performed at a fixed frequency during the exploration phase of the agent. The update of the parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}^{\boldsymbol{Q}}, \boldsymbol{\theta}^{\pi}, \boldsymbol{\theta}^{\boldsymbol{Q}'}, \boldsymbol{\theta}^{\pi'})$ follows the subsequent steps: first, we randomly retrieve n samples from the RB, where n represents the chosen batch size. For each experience i only the new state $\boldsymbol{x}_{i+1}$ is passed to the actor copy network $\boldsymbol{\pi}'$ which returns the action $\boldsymbol{u}_{i+1}$. The pair $(\boldsymbol{u}_{i+1}, \boldsymbol{x}_{i+1})$ is used as input to $\boldsymbol{Q}'$ that estimates the associated Q-value. The obtained result is then combined with the sampled experience reward r_i to compute the temporal difference, which is defined as

$$y_i = r_i + \lambda\, \boldsymbol{Q}'(\boldsymbol{x}_{i+1}, \boldsymbol{\pi}'(\boldsymbol{x}_{i+1}|\boldsymbol{\theta}^{\pi'})|\boldsymbol{\theta}^{\boldsymbol{Q}'}),$$

where λ is the discount factor. The final loss L, calculated by subtracting to y_i the $\boldsymbol{Q}$ function computed over the sampled values $(\boldsymbol{u}_i, \boldsymbol{x}_i)$ as

$$L = \frac{1}{n}\sum_i (y_i - \boldsymbol{Q}(\boldsymbol{x}_i, \boldsymbol{u}_i|\boldsymbol{\theta}^{\boldsymbol{Q}})^2),$$

is then used to retrieve the gradient for the critic $\boldsymbol{Q}$.

Finally the actor gradient $\boldsymbol{\pi}$ can be approximated as

$$\nabla_{\boldsymbol{\theta}^{\pi}}\boldsymbol{\pi} \approx \frac{1}{n}\sum_i \nabla_{\boldsymbol{u}}\boldsymbol{Q}(\boldsymbol{x}, \boldsymbol{u}|\boldsymbol{\theta}^{\boldsymbol{Q}})|_{\boldsymbol{x}_i, \boldsymbol{\pi}(\boldsymbol{x}_i)} \nabla_{\boldsymbol{\theta}^{\pi}}\boldsymbol{\pi}(\boldsymbol{x}|\boldsymbol{\theta}^{\pi})|_{\boldsymbol{x}_i}.$$

Each gradient is then back-propagated to update the values of the parameters $\boldsymbol{\theta}$ with two different learning rates α_a, α_c (see 2.19 of Ch. 2), respectively for the actor and for the critic. At the end of the training phase, the control policy $\boldsymbol{\pi}$ is obtained. It will be used online to generate the reference trajectory for the NMPC.

During the offline policy learning phase, constraints cannot be explicitly enforced, but their violation can only be penalized through the reward function.

6.2.2 NMPC for Online Constraints Satisfaction

NMPC is an advanced control method that uses a nonlinear dynamic model of a system to define a finite-time constrained NLP that is solved numerically in an iterative fashion. Let

$$\mathcal{T}(\boldsymbol{x}_k) = \left\{(\boldsymbol{u}_k^{\text{ref}}, \boldsymbol{x}_k^{\text{ref}}), k = 0, \dots, N\right\}$$

represents the control-state reference trajectory where $\boldsymbol{x}_k^{\text{ref}}$ and $\boldsymbol{u}_k^{\text{ref}} = \boldsymbol{\pi}(\boldsymbol{x}_k^{\text{ref}})$ are the state and control input at k generated by forward integrating the simulation robot model with $\boldsymbol{\pi}(\cdot)$ from the initial condition $\boldsymbol{x}_k$. We formulate the NLP as

$$\min_{\substack{\boldsymbol{u}_0,\dots,\boldsymbol{u}_{N-1} \\ \boldsymbol{s}_0,\dots,\boldsymbol{s}_N}} \sum_{i=0}^{N-1} J_s(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{s}_i) + J_N(\boldsymbol{x}_N)$$

subject to

$$\begin{aligned} \boldsymbol{x}_{i+1} - \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) &= 0, & i &= 0 \dots, N-1, \\ \boldsymbol{g}(\boldsymbol{x}_i) - \boldsymbol{s}_i &\leq 0, & i &= 1, \dots, N, \\ \boldsymbol{h}(\boldsymbol{u}_i) &\leq 0, & i &= 0, \dots, N-1, \end{aligned}$$

with $\boldsymbol{x}_0 = \boldsymbol{x}_k$ again the actual state of the robot. Differently from the optimization of Sec. 4.3.3 and Sec. 5.3.1, this formulation contains a second set of variables $\boldsymbol{s}$, called *slack variables*, that can be manipulated by the solver in order to perform constraints relaxation if the problem became unfeasible, i.e. no solution exists that respect the actual constraints. The cost terms take the form

$$J_s(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{s}_i) = (\boldsymbol{x}_i^{\text{ref}} - \boldsymbol{x}_i)^\top \boldsymbol{Q} (\boldsymbol{x}_i^{\text{ref}} - \boldsymbol{x}_i) + (\boldsymbol{u}_i^{\text{ref}} - \boldsymbol{u}_i)^\top \boldsymbol{R} (\boldsymbol{u}_i^{\text{ref}} - \boldsymbol{u}_i) + \frac{\rho}{2} \boldsymbol{s}_i^2,$$

$$J_N(\boldsymbol{x}_N) = (\boldsymbol{x}_i^{\text{ref}} - \boldsymbol{x}_N)^\top \boldsymbol{Q}_N (\boldsymbol{x}_i^{\text{ref}} - \boldsymbol{x}_N)$$

where the last term in $J_s(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{s}_i)$ is present to minimize as much as possible constraints relaxation. In fact, one should choose the weight ρ in order to make this term more relevant with respect to the others to allow constraints violation only when became strictly necessary.

It is important to notice that each time the system reaches a new state $\boldsymbol{x}_{k+1}$, before solving the NMPC problem, an update of the reference control-state trajectory is required. Hence, to generate the new trajectory, the policy $\boldsymbol{\pi}$ learned offline is applied to the simulated robot. The resulting control-state trajectory $\mathcal{T}(\boldsymbol{x}_{k+1})$ is used as a reference in the cost function of the NMPC problem. Updating the reference trajectory plays a central role in the proposed algorithm. Thus, in the

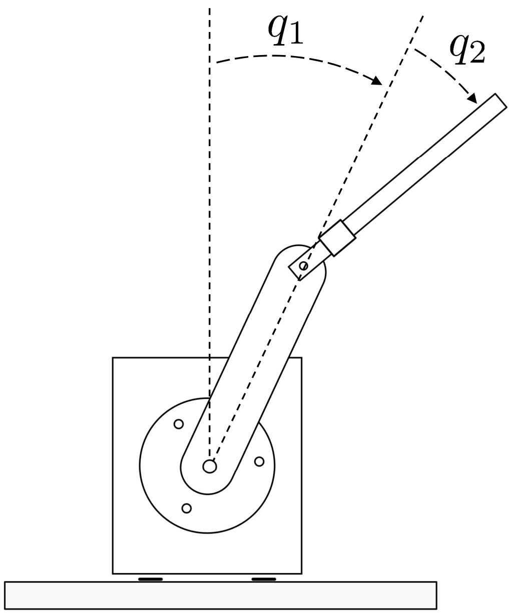

Figure 6.3. Schematic of the Pendubot coordinate system.

event that the current reference trajectory violates the constraints, the NMPC will drive the system away from the current reference trajectory, which will become obsolete.

In real-time applications, the NMPC needs to be solved at every sampling instant under tight runtime requirements. For this reason, here we employed the RTI scheme in order to obtain a real-time control law.

6.3 Results

We tested the proposed controller on the Pendubot (Fig. 6.1), an underactuated, planar, robotic system described previously in Ch. 5. We consider again the problem of swinging up and balancing the robot around the up-up equilibrium starting from the stable down-down equilibrium, while satisfying the imposed velocity constraints. Differently from what is done in Ch. 5, in order to be compliant with our published paper [20] and its related video, we define a coordinate system where the desired final state is $\boldsymbol{x}_g = (q_1, q_2, \dot{q}_1, \dot{q}_2) = (0, 0, 0, 0)$, with q_i and $\dot{q}_i$ respectively the position and the velocity of the i-th joint (Fig. 6.3). The joint position q_1 is measured with respect to the upwards vertical axis, the angle q_2 is defined relatively to the first joint, while the start configuration is defined as $\boldsymbol{x}_s = (q_1, q_2, \dot{q}_1, \dot{q}_2) = (-\pi, 0, 0, 0)$.

In the following we will describe the setup of the learning and control phases. Moreover, we will show the behaviour of the Pendubot under two different constraint settings, either in simulation and on the real platform. Finally, we will show that the continuous update of the reference trajectory is critical to successfully achieve the desired task.

6.3.1 Offline Learning Phase

The learning of the control policy $\boldsymbol{\pi}$ is performed offline using the simulated robot model described in the published version of our work [20]. Its derivation, with the different set of generalized coordinates used in Ch. 5, can be found in the Appendix A.1. In order to train $\boldsymbol{\pi}$, a reward function must be defined. The function used here penalizes the distance between the current state and the desired one $\boldsymbol{x}_g = (0, 0, 0, 0)$. Thus, we define the reward $r(\boldsymbol{x})$ as

$$r(\boldsymbol{x}) = -r_1(\boldsymbol{q}) - \alpha_v r_2(\dot{\boldsymbol{q}}) - \alpha_\tau |\tau|, \tag{6.1}$$

where $\alpha_v = 0.1$, $\alpha_\tau = 0.01$ are weighting parameters, and $r_1(\boldsymbol{q}) = |q_1| + |q_2|$. Moreover, as it is desired to have an upper limit on the joint velocities, both in experiments and in simulations, the quantity $r_2(\dot{\boldsymbol{q}}) = |\dot{q}_1| + |\dot{q}_2|$ is introduced in eq. (6.1). Finally, the term $-\alpha_\tau|\tau|$ is used to minimize the control effort, with τ representing the applied torque.

We optimize the control policy $\boldsymbol{\pi}$ on an Intel i7-4770 processor with a maximum frequency of 3.40 GHz. The hyperparameters of DDPG were tuned by trial and error performing different learning experiments, and were finally chosen as following

- learning late α_a actor: 0.0001
- learning late α_c critic: 0.0001
- discount factor λ: 0.9
- target network copy rate α_t: 0.01
- layers Actor: 3
- neurons Actor: [80, 64, 1]
- layers Critic: 4
- neurons Critic: [100, 60, 60, 1]

Furthermore, all the first and hidden layers were composed by a relu activation function (see Sec. 2.1.2), while the output layers were composed in the case of the Critic by a linear activation, while for the Actor by a tanh postmultiplied by a scalar. Usually the use of a tanh is a common practice in RL since it permits to obtain a *bounded* control action, meaning that it cannot exceed a maximum chosen value. In our Pendubot prototype, in fact the maximum applicable torque is 0.4 Nm. During training, just after 500 episodes which lasted approximately 1 hour, the learned policy $\boldsymbol{\pi}$ was able to conclude the task performing both the swing-up and the balancing. Moreover, we trained the system exploiting only one core of our CPU, but usually one can cut considerably the learning time by performing a parallel training of the agent exploiting all the CPU cores available. More recently, different physics simulators such as [131, 132] permit to exploit even the cores of the graphics card (GPU), without sharing any data with the processor during training and hence reducing greatly all the communication bottlenecks. For example, thanks to this idea in [133] more than 4000 robots were trained in parallel on a single workstation with just an Nvidia GPU.

6.3.2 Online Control Phase

The Pendubot system is characterized by highly nonlinear and fast dynamics, and therefore a high frequency controller is necessary to reach the desired goal. The proposed framework runs in real-time with a control frequency of 500 Hz. To solve the NMPC for both the simulations and the experiments we used the `ACADO Toolkit` [73], with a fixed prediction horizon of $N = 10$, obtaining a mean compute time of 98.6 μs. We used an ERK4 integration scheme to discretize the nonlinear dynamics of the system, while the weighting matrices and the slack penalty in the cost function of the optimization problem were chosen as $\boldsymbol{Q} = diag(3, 3, 1, 1)$, $R = 5$, $\boldsymbol{Q}_N = diag(3, 3, 1, 1)$, and $\rho = 40$. The simulated robot model defined in [20] is used both within the NMPC optimization problem and for generating the desired reference trajectory querying the policy $\boldsymbol{\pi}$ at each time step, which requires about 1.2 ms. The swing-up of the Pendubot can be performed either rapidly or with an energy pumping maneuver, reaching the desired state $\boldsymbol{x}_g$ through an oscillatory motion. In both cases, when the state is close to $\boldsymbol{x}_g$, a local controller, usually based on a linearization of the system around the final equilibrium point, can be used for the balancing phase. To this end we implemented an LQR controller (3.1) for the experiment on the real platform, using the same weighting matrices $\boldsymbol{Q}$,$\boldsymbol{Q}_N$ and $\boldsymbol{R}$.

Since the robot is only equipped with encoders, angular velocities are numerically derived from position measures. Still, as explained in Ch. 5, given the high resolution of the robot sensors we obtain a good estimation of $\dot{\boldsymbol{q}}$ without the need of any filtering.

6.3.3 Simulation and Experimental Results

In this section we show the results obtained in simulation and on the real robot, under two different constraint settings. The control policy, learned as explained in Sec. 6.2.1, is able to solve the swing-up including the balancing phase. The maximum velocities reached in simulation during the motion are 9.2 rad/sec and 9.8 rad/sec for the first and second joint, respectively. For clarity, in Figs. 6.4–6.5–6.6, and Fig. 6.7, we represent with blue lines the solution obtained with the NMPC, and with the red dashed lines the state trajectory $\boldsymbol{q}_i^{\boldsymbol{\pi}}$ obtained using only the control policy $\boldsymbol{\pi}$.

In the first setting, we impose a symmetric velocity constraint $\dot{q}_{1,\mathrm{b}} = 7.2$ rad/sec on the first joint and a maximum admissible torque $\tau_{1,\mathrm{b}} = 0.4$ N $\cdot$ m. We compare our approach with an NMPC where the reference trajectory obtained by the control policy is never updated. In our case (Fig. 6.4), due to the imposed constraints the proposed controller steers the system to new feasible states, forcing the policy to recalculate a different trajectory, while successfully reaching the up-up configuration. On the other hand, if an NMPC with a fixed reference trajectory is used, the robot does not reach the desired final configuration as shown in Fig. 6.5. It is therefore clear from these results, that recomputing the control-state trajectory at each time

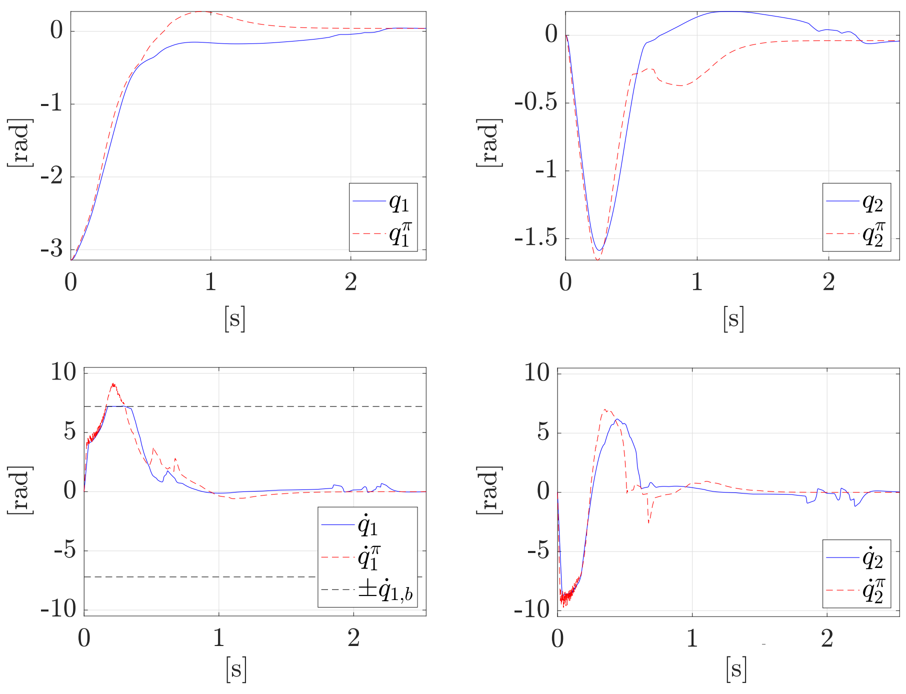

Figure 6.4. Simulation result obtained with the constraints $\dot{q}_{1,\mathrm{b}} = 7.2$ rad/sec with our approach.

step is crucial to successfully complete the desired task.

In the next simulation, we define a second constraints setting, with $\dot{q}_{1,\mathrm{b}} = 7.8$ rad/sec, $\tau_{1,\mathrm{b}} = 0.4$ N $\cdot$ m and $\dot{q}_{2,\mathrm{b}} = 5$ rad/sec for the angular velocity of the second joint. As shown in Fig. 6.6, our controller is still able to perform the desired task. The maximum admissible velocity for the second joint is quickly reached, and the final motion of the system has a larger change with respect to the solution obtained with the first constraints setting.

Finally, we validated our approach on the real Pendubot system using a second constraints scenario. Fig. 6.7 shows that our method is able to successfully perform the swing-up maneuver despite the velocity constraints. In this experiment a final LQR is used to stabilize the system around the equilibrium point, and the use of soft constraints in the NMPC formulation becomes necessary to avoid infeasibility due to model inaccuracies. Indeed, the introduction of slack variables determines small constraint violations of 0.1 rad/s and 1 rad/s, respectively on the first and second joint velocities (see Fig. 6.7). Controlling the real robot, in fact, involves additional challenges due to model uncertainties, since both the NMPC and the policy are unaware of the true dynamical model of the system. Hence, $\boldsymbol{\pi}$ will not generate exactly a dynamically feasible trajectory, and the use of soft constraints in the NMPC formulation becomes necessary to recover the local feasibility of the optimization problem. Although this approach does not necessarily guarantee that the constraints will be exactly satisfied, it does allow a new control input to be computed that will try to minimize the violation as much as possible. Thus, the oscillatory behaviour, that is visible on the velocity of the second joint (Fig. 6.7), depends on the effect

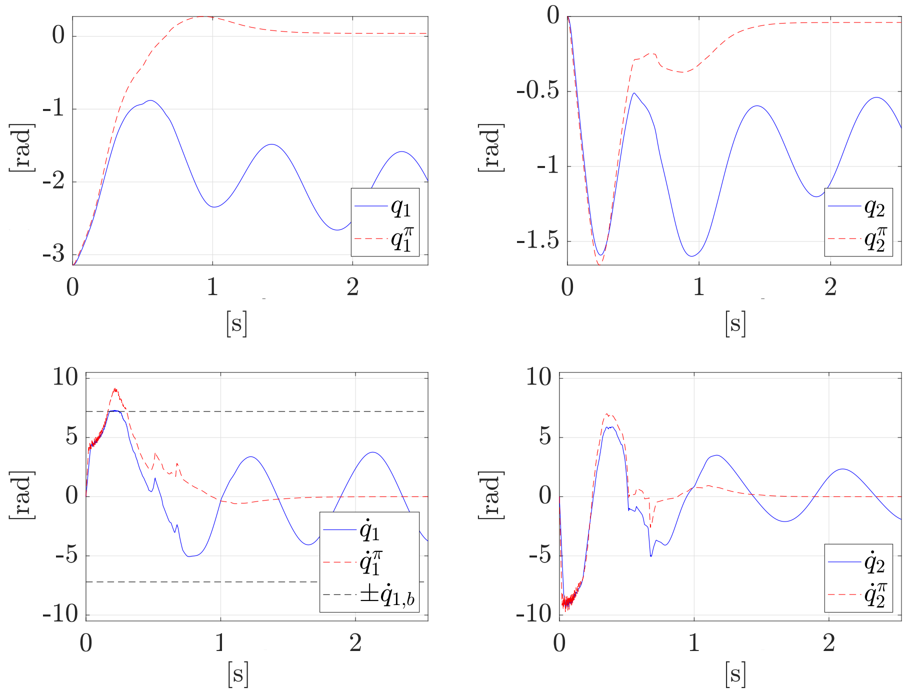

Figure 6.5. Simulation result obtained with the constraints $\dot{q}_{1,\mathrm{b}} = 7.2$ rad/sec using a NMPC with a fixed trajectory. Given the presence of underactuation, the fixed trajectory mechanism does not allow the completion of the task.

of the cost minimization associated with the constraints violation. Model and parameters uncertainties, due to mechanical wear and time-varying dynamics (like the dynamic friction induced by the sliding contacts for the encoder of link 2), explain the difference between the results obtained in simulation and on the real robot.

A video of the proposed experiments can be found at the following link `https://www.youtu.be/5KATfbDwKlI`.

6.4 Chapter Summary

In this chapter we have presented an approach for imposing constraints to a learned control policy. We test the proposed controller both in simulation and on a real Pendubot robot, showing that the continuous interplay between the NMPC and the learned policy is at the base of the constraints enforcement.

In the future, we plan to extensively study the NMPC steering capabilities. Not every constraint can be arbitrarily imposed in the optimization problem, since no assumption is made over the policy capability $\boldsymbol{\pi}$ to generate a satisfactory trajectory in the feasible region. A possible solution will be to automatically retrieve an index that measures the ability of the policy to accomplish the given task from any state, relaxing the constraints when it is needed using the slack variables. A second extension will be the improvement of the robustness over model uncertainties. Multiple policies can be learned in simulation with different dynamical parameters,

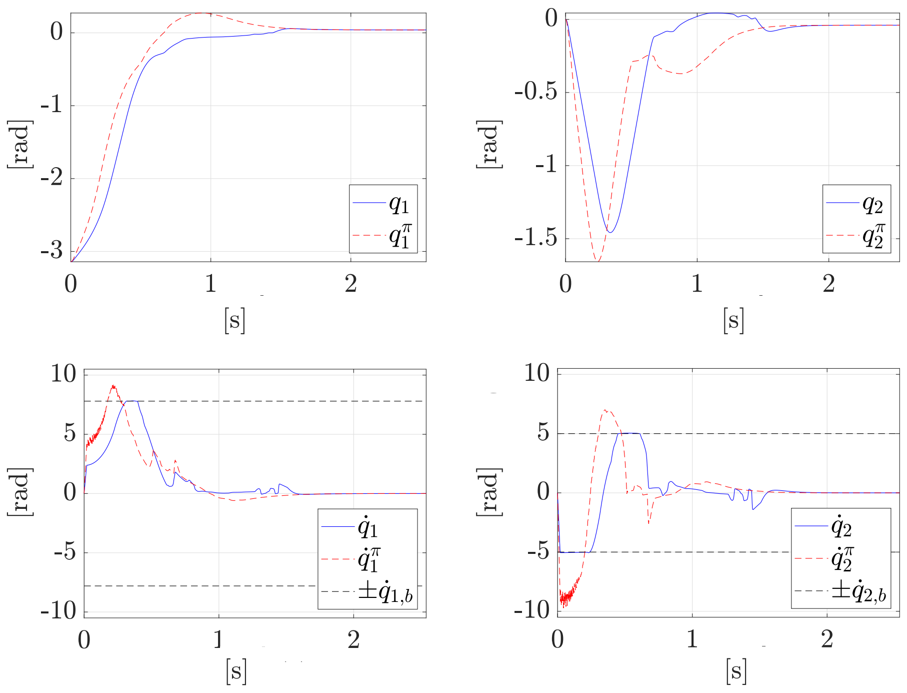

Figure 6.6. Simulation result obtained with the constraints $\dot{q}_{1,\mathrm{b}} = 7.2$ rad/sec and $\dot{q}_{2,\mathrm{b}} = 5$ rad/sec using our approach.

and the choice of the appropriate $\boldsymbol{\pi}$ can be recomputed online within the NMPC formulation.

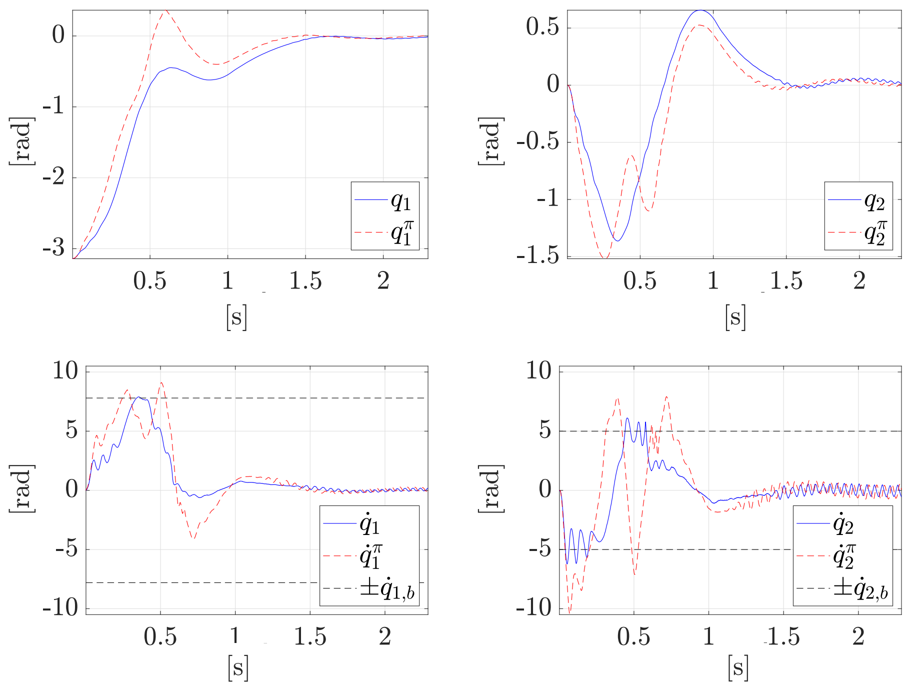

Figure 6.7. Experimental result obtained with the constraints $\dot{q}_{1,\mathrm{b}} = 7.8$ rad/sec, $\dot{q}_{2,\mathrm{b}} = 5$ rad/sec. Constraints violations can be observed during motion due to the presence of model uncertainties.

Chapter 7

Conclusions

In this thesis we have presented three different learning-based techniques linked to the concepts of model and control learning, applicable both in the case of fully actuated and underactuated robots.

We started this manuscript by presenting some background materials on the field of Machine Learning, describing some regression techniques (Gaussian Process and Neural Network) that can be used for learning the dynamical model of the system or part of it. In the same chapter, we described Reinforcement Learning, one of the most promising techniques to learn a controller just from data, highlighting one of its most important drawbacks that is the impossibility to enforce hard constraints.

We then moved to the field of Optimal Control, since most of the controllers used in the rest of the thesis are related to this branch of Control Theory. We described the Linear Quadratic Regulator, used for balancing the Pendubot in two of our works, and then we illustrated Model Predictive Control, both in the linear and nonlinear case, which is one of the few control laws able to enforce constraints to robot motion.

Finally, in the last three chapters of this thesis, we presented our contributions in the literature. Specifically, in Ch. 4 we have derived a feedback linearization controller that is able to reconstruct during motion, thus online, model uncertainties, obtaining superior performance due to a batter cancellation of the system nonlinearities. The proposed controller is composed of a high level linear Model Predictive Controller, applied on a simple double integrator model which is able to deal with constraints both on the inputs and on the states, plus a learned correction term reconstructed using the Controllability Gramian. To prove the effectiveness of our method, we tested the proposed controller in simulation on the 7-dofs Kuka LBR robot.

In Ch. 5 we have extended the previous method in the case of underactuated robots, systems that require specialized controllers in order to deal with the absence of control inputs. In this case, we showed that due to model uncertainties it is necessary not only to increase the performance of the controller but even to plan a dynamically feasible trajectory, a strong requirement for an underactuated system that cannot be met in the presence of uncertain dynamics. We resolved this problem with an iterative algorithm that alternates a planning phase, solved by an SQP method and where uncertainties on the unactuated part of the system are taken into consideration, and a control phase where a simple PD control law plus a learned correction term is employed to perform stable tracking. We have shown that in just a few iterations our method is able to solve different tasks, no matter how uncertain is its dynamical model and independently from the dimension of the robot state space. We tested the proposed approach both on the Pendubot, a 2R underactuated system, and on a similar 3R underactuated manipulator with two passive joints. Furthermore, we conducted extensive experiments in order to show the capabilities of our method in a realistic scenario.

Finally, in Ch. 6 we have tackled the problem of constraints satisfaction in Reinforcement Learning, which has we detailed cannot in general be hardly imposed neither during training nor during test using standard learning techniques. We proposed to this end a shielding mechanism composed of a fast Nonlinear Model Predictive Control implementation (the Real Time Iteration scheme), which is able to deviate the system from unfeasible states continuously querying the learned control policy in order to obtain an up-to-date trajectory. In this way, we were able to satisfy different imposed constraints at demand on the Pendubot, both in simulation and on the real robot.

Appendix A

Dynamic Model

The dynamics of a robot can be derived using the Lagrange formulation, starting from the choice of a set of variables $\boldsymbol{q}$, called the *generalized coordinates*, to describe all the link positions of a manipulator. In order to compute the *lagrangian*, defined as

$$\mathcal{L} = T - U \tag{A.1}$$

we need first to compute the kinetic energy T and the potential energy U of the robot. For a single rigid body, T is composed by two terms

$$T = \frac{1}{2} m \boldsymbol{v}_c^T \boldsymbol{v}_c + \frac{1}{2} I \dot{q}^2 \tag{A.2}$$

where m, $\boldsymbol{v}_c$ are respectively the mass of the link and the velocity of its center of mass and I the body inertia. The first term in eq. (A.2) represents the absolute velocity of the center of the mass (CoM), while the second is the absolute velocity of the whole link. The potential energy U instead can be written as

$$U = -m\boldsymbol{g}^\top \boldsymbol{p}$$

where $\boldsymbol{g} = (0, 0, -g)$ is the gravity vector with g the force of gravity, and $\boldsymbol{p}$ is the position of the CoM. Once we have written the energy of the system, we can proceed computing the *euler-lagrange equations*, which are defined as following

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = \xi_i \quad i = 1, \dots, n \tag{A.3}$$

where n is the number of joints and ξ_i groups the generalized forces acting on the system, e.g. torque and friction. The final model of a manipulator, after computing the energy of the robot and eq. (A.3), will assume the general form

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} - \boldsymbol{\tau}_f(\boldsymbol{q}, \dot{\boldsymbol{q}}), \tag{A.4}$$

where $\boldsymbol{M}$ is the mass matrix, that depends only on the generalized coordinates, $\boldsymbol{c}$ is the summation of the Coriolis and centrifugal forces while $\boldsymbol{g}$ is the gravity vector. Finally, $\boldsymbol{\tau}_f(\boldsymbol{q}, \dot{\boldsymbol{q}})$ comprises the friction forces that can be dependant both on the position and on the velocity of the chosen coordinates (*viscous friction*) or just proportional to their velocity (*static friction*), and $\boldsymbol{\tau}$ is the torque vector which can have some zero elements inside. If the torque inputs are less than the generalized coordinates we talk about underactuated systems, otherwise the system is fully-actuated or even over-actuated, e.g. as in the case of a redundant manipulator such as the Kuka LBR iiwa used as a testbench in Ch. 4.

A.1 Pendubot model

The Pendubot is a 2R underactuated robot, with the first joint actuated while the second passive. We can write the dynamical model choosing as generalized coordinates $\boldsymbol{q} = (q_1, q_2)$ represented in Fig. 5.3, with q_1 taken with respect to the downwards vertical axis while q_2 relative to the first joint. The kinetic energy of the system takes the form of

$$\begin{aligned} T_1 &= \frac{1}{2} I_1 \dot{q}_1^2 \\ T_2 &= \frac{1}{2} m_2 \boldsymbol{v}_{c2}^T \boldsymbol{v}_{c2} + \frac{1}{2} I_2 \dot{q}_2^2 \end{aligned}$$

where I_i,m_i and l_i are, respectively, the moment of inertia, the mass and the length of the i-th link. Furthermore, $\boldsymbol{v}_{c2}$ is derivative of the position $\boldsymbol{p}_{c2}$ of the center of mass of the second link, equal to

$$\begin{aligned} \boldsymbol{p}_{c2} &= \begin{pmatrix} l_1 \sin(q_1) + d_2 \sin(q_2 + q_1) \\ -l_1 \cos(q_1) - d_2 \cos(q_2 + q_1) \end{pmatrix} \\ \boldsymbol{v}_{c2} &= \begin{pmatrix} l_1 \cos(q_1)\dot{q}_1 + d_2 \cos(q_2 + q_1)(\dot{q}_2 + \dot{q}_1) \\ l_1 \sin(q_1)\dot{q}_1 + d_2 \sin(q_2 + q_1)(\dot{q}_2 + \dot{q}_1) \end{pmatrix} \end{aligned}$$

where d_i is the distance of the center of mass of the i-th link from the center of the i-th joint, while the potential energy is instead

$$\begin{aligned} U_1 &= -m_1 g d_1 \cos(q_1) \\ U_2 &= -m_2 g \left(l_1 \cos(q_1) + d_2 \cos(q_2) \right) \end{aligned}$$

Computing the euler-lagrange equations (A.3) we finally arrive to the Pendubot dynamical model below

$$\boldsymbol{M}(\boldsymbol{q}) = \begin{pmatrix} a_1 + 2a_2 c_2 & a_3 + a_2 c_2 \\ a_3 + a_2 c_2 & a_3 \end{pmatrix} \quad \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{pmatrix} a_2 s_2 \dot{q}_2 (\dot{q}_2 + 2\dot{q}_1) \\ a_2 s_2 \dot{q}_1^2 \end{pmatrix}$$
$$\boldsymbol{g}(\boldsymbol{q}) = \begin{pmatrix} a_4 s_1 + a_5 s_{12} \\ a_5 s_{12} \end{pmatrix}$$

where we have set $s_i = \sin(q_i)$, $c_i = \cos(q_i)$ and $s_{ij} = \sin(q_i + q_j)$ for simplicity, and where the a_i coefficients that appear in the dynamical model above are

$$\begin{aligned} a_1 &= I_1 + m_1 d_1^2 + I_2 + m_2 \left(l_1^2 + d_2^2 \right) \\ a_2 &= m_2 l_1 d_2 \\ a_3 &= I_2 + m_2 d_2^2 \\ a_4 &= g \left(m_1 d_1 + m_2 l_1 \right) \\ a_5 &= g m_2 d_2 \end{aligned}$$

Finally, the dynamic and kinematic parameters of the Pendubot are listed in Table A.1.

Table A.1. Physical parameters for the Pendubot

Parameter	*Value*	*Description*
m_1	0.193 kg	total mass of link 1
l_1	0.1483 m	length of link 1
d_1	0.1032 m	distance to the center of mass of link 1
I_1	$3.54 \cdot 10^{-4}$ kg·m^2	inertia moment of link 1
m_2	0.073 kg	total mass of link 2
l_2	0.1804 m	length of link 2
d_2	0.1065 m	distance to the center of mass of link 2
I_2	$2 \cdot 10^{-4}$ kg·m^2	inertia moment of link 2

Bibliography

[1] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[2] O. Kroemer, R. Detry, J. H. Piater, and J. Peters, “Combining active learning and reactive control for robot grasping,” *Robotics and Autonomous Systems*, vol. 58, pp. 1105–1116, 2010.

[3] S. Joshi, S. Kumra, and F. Sahin, “Robotic grasping using deep reinforcement learning,” in *Proceedings IEEE 16th International Conference on Automation Science and Engineering*, 2020, pp. 1461–1466.

[4] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, “Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning,” in *Proceedings IEEE International Conference on Robotics and Automation*, 2018, pp. 5113–5120.

[5] C. Gaskett, L. Fletcher, and A. Zelinsky, “Reinforcement learning for a vision based mobile robot,” in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, 2000, pp. 403–409.

[6] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, “Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4298–4305, 2019.

[7] H. Kimura, T. Yamashita, and S. Kobayashi, “Reinforcement learning of walking behavior for a four-legged robot,” in *Proceedings 40th IEEE Conference on Decision and Control*, vol. 1, 2001, pp. 411–416.

[8] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng, “Learning cpg-based biped locomotion with a policy gradient method: Application to a humanoid robot,” *The International Journal of Robotics Research*, vol. 27, pp. 213 – 228, 2008.

[9] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, “Learning agile and dynamic motor skills for legged robots,” *Science Robotics*, vol. 4, 2019.

[10] C. Paduraru, D. J. Mankowitz, G. Dulac-Arnold, J. Z. Li, N. Levine, S. Gowal, and T. Hester, “Challenges of real-world reinforcement learning: definitions, benchmarks and analysis,” *Machine Learning*, vol. 110, pp. 2419–2468, 2021.

[11] V. Utkin, "Sliding mode control design principles and applications to electric drives," *IEEE Transactions on Industrial Electronics*, vol. 40, no. 1, pp. 23–36, 1993.

[12] S. S. Sastry and M. Bodson, *Adaptive Control: Stability, Convergence and Robustness*. Prentice-Hall, 1989.

[13] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," in *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, 2020, p. 269–296.

[14] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin, "Learning quadrotor dynamics using neural network for flight control," in *Proceeding IEEE 55th Conference on Decision and Control*, 2016, pp. 4653–4660.

[15] A. Punjani and P. Abbeel, "Deep learning helicopter dynamics models," in *Proceeding IEEE International Conference on Robotics and Automation*, 2015, pp. 3223–3230.

[16] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, "Learning-based model predictive control for autonomous racing," *IEEE Robotics and Automation Letters*, vol. 4, pp. 3363–3370, 2019.

[17] V. Kumar, E. Todorov, and S. Levine, "Optimal control with learned local models: Application to dexterous manipulation," *Proceeding IEEE International Conference on Robotics and Automation*, pp. 378–383, 2016.

[18] M. Capotondi, G. Turrisi, C. Gaz, V. Modugno, G. Oriolo, and A. De Luca, "An online learning procedure for feedback linearization control without torque measurements," in *Proceedings Machine Learning Research (3rd Conference on Robot Learning)*, vol. 100, 2020, pp. 1359–1368.

[19] G. Turrisi, M. Capotondi, C. Gaz, V. Modugno, G. Oriolo, and A. De Luca, "On-line learning for planning and control of underactuated robots with uncertain dynamics," *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 358–365, 2022.

[20] G. Turrisi, B. Barros Carlos, M. Cefalo, V. Modugno, L. Lanari, and G. Oriolo, "Enforcing constraints over learned policies via nonlinear MPC: Application to the Pendubot," *IFAC PapersOnLine*, vol. 53, no. 2, pp. 9502–9507, 2020.

[21] A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin, "Provably safe and robust learning-based model predictive control," *Automatica*, vol. 49, no. 5, pp. 1216–1226, 2013.

[22] C. D. McKinnon and A. P. Schoellig, "Learning probabilistic models for safe predictive control in unknown environments," in *Proceedings 18th European Control Conference*, 2019, pp. 2472–2479.

[23] J. Nubert, J. Köhler, V. Berenz, F. Allgöwer, and S. Trimpe, "Safe and fast tracking on a robot manipulator: Robust mpc and neural network control," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3050–3057, 2020.

[24] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural lander: Stable drone landing control using learned dynamics," in *Proceedings International Conference on Robotics and Automation*, 2019, pp. 9784–9790.

[25] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for Machine Learning.* Cambridge University Press, 2020.

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

[27] M. Seeger, C. Williams, and N. Lawrence, "Fast forward selection to speed up sparse Gaussian process regression," in *Proceedings 9th International Work. on Artificial Intelligence and Statistics*, 2003.

[28] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious model predictive control using gaussian process regression," *IEEE Transaction on Control Systems Technology*, vol. 28, no. 6, pp. 2736–2743, 2020.

[29] T. Beckers, J. Umlauft, D. Kulic, and S. Hirche, "Stable gaussian process based tracking control of lagrangian systems," in *Proceedings IEEE 56th Annual Conference on Decision and Control*, 2017, pp. 5180–5185.

[30] G. da Silva Lima, S. Trimpe, and W. M. Bessa, "Sliding mode control with gaussian process regression for underwater robots," *Journal of Intelligent and Robotic Systems*, pp. 1–12, 2020.

[31] Y. Pan and E. A. Theodorou, "Data-driven differential dynamic programming using gaussian processes," in *Proceedings American Control Conference*, 2015, pp. 4467–4472.

[32] A. Gahlawat, P. Zhao, A. Patterson, N. Hovakimyan, and E. A. Theodorou, "L1-gp: L1 adaptive control with bayesian learning," in *Learning for Dynamics and Control*, 2020.

[33] A. Capone and S. Hirche, "Backstepping for partially unknown nonlinear systems using gaussian processes," *IEEE Control Systems Letters*, vol. 3, no. 2, pp. 416–421, 2019.

[34] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016.

[36] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, "Deep neural networks for improved, impromptu trajectory tracking of quadrotors," in *Proceedings IEEE International Conference on Robotics and Automation*, 2017, pp. 5183–5189.

[37] F. Cursi, V. Modugno, L. Lanari, G. Oriolo, and P. Kormushev, "Bayesian neural network modeling and hierarchical mpc for a tendon-driven surgical robot with uncertainty minimization," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2642–2649, 2021.

[38] A. J. Taylor, A. W. Singletary, Y. Yue, and A. D. Ames, "Learning for safety-critical control with control barrier functions," in *Learning for Dynamics and Control*, 2020.

[39] M. Lutter, C. Ritter, and J. Peters, "Deep lagrangian networks: Using physics as model prior for deep learning," in *Proceedings International Conference on Learning Representations*, 2019.

[40] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[41] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016.

[42] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, and D. Silver, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, pp. 350–354, 2019.

[43] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, pp. 3–20, 2018.

[44] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, 2020.

[45] Z. Xie, G. Berseth, P. Clary, J. W. Hurst, and M. van de Panne, "Feedback control for cassie with deep reinforcement learning," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018, pp. 1241–1246.

[46] D. Bertsekas, *Dynamic Programming and Optimal Control.* Athena scientific, 2017, vol. 1.

[47] D. Q. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal of Control*, vol. 3, pp. 85–95, 1966.

[48] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *Proceedings International Conference on Informatics in Control, Automation and Robotics*, 2004.

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

[50] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," 1994.

[51] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 2004.

[52] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *SIGART Bulletin*, vol. 2, pp. 160–163, 1991.

[53] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[54] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 2004.

[55] D. Silver, G. Lever, N. M. O. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings International Conference on Machine Learning*, 2014.

[56] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber, "Policy gradients with parameter-based exploration for control," in *International Conference on Artificial Neural Networks*, 2008.

[57] S. Levine and V. Koltun, "Guided policy search," in *ICML*, 2013.

[58] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Foundation and Trends in Robotics*, vol. 2, pp. 1–142, 2013.

[59] R. Tedrake, T. W. Zhang, and H. S. Seung, "Stochastic policy gradient reinforcement learning on a simple 3d biped," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 2004, pp. 2849–2854.

[60] H. Durrant-Whyte, N. Roy, and P. Abbeel, *Learning to Control a Low-Cost Manipulator Using Data-Efficient Reinforcement Learning.* MIT Press, 2012.

[61] Y. Song and D. Scaramuzza, "Policy search for model predictive control with application to agile drone flight," *to appear in IEEE Transaction on Robotics*, 2022.

[62] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” in *Proceedings 4th International Conference on Learning Representations*, 2016.

[63] V. Klemm, A. Morra, L. Gulich, D. Mannhart, D. Rohr, M. Kamel, Y. de Viragh, and R. Siegwart, “Lqr-assisted whole-body control of a wheeled bipedal robot with kinematic loops,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3745–3752, 2020.

[64] A. Marco, P. Hennig, J. Bohg, S. Schaal, and S. Trimpe, “Automatic lqr tuning based on gaussian process global optimization,” in *Proceedings IEEE International Conference on Robotics and Automation*, 2016, pp. 270–277.

[65] R. Tedrake, I. R. Manchester, M. M. Tobenkin, and J. W. Roberts, “Lqr-trees: Feedback motion planning via sums-of-squares verification,” *The International Journal of Robotics Research*, vol. 29, pp. 1038 – 1052, 2010.

[66] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 2nd ed. Nob Hill Pub., 2017.

[67] D. Bruder, B. Gillespie, C. D. Remy, and R. Vasudevan, “Modeling and control of soft robots using the koopman operator and model predictive control,” *ArXiv*, 2019.

[68] P.-b. Wieber, “Trajectory free linear model predictive control for stable walking in the presence of strong perturbations,” in *2006 6th IEEE-RAS International Conference on Humanoid Robots*, 2006, pp. 137–142.

[69] H. Michalska and D. Mayne, “Robust receding horizon control of constrained nonlinear systems,” *IEEE Transactions on Automatic Control*, vol. 38, no. 11, pp. 1623–1633, 1993.

[70] L. Chisci, J. A. Rossiter, and G. Zappa, “Systems with persistent disturbances: predictive control with restricted constraints,” *Automatica*, vol. 37, pp. 1019–1028, 2001.

[71] J. Nocedal and S. J. Wright, *Numerical Optimization.* Springer, 1999.

[72] M. Diehl, H. G. Bock, and J. P. Schlöder, “A real-time iteration scheme for nonlinear optimization in optimal feedback control,” *SIAM Journal on Control and Optimization*, vol. 43, no. 5, pp. 1714–1736, 2005.

[73] B. Houska, H. J. Ferreau, and M. Diehl, “ACADO toolkit - an open-source framework for automatic control and dynamic optimization,” *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[74] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, “acados: a modular open-source framework for fast embedded optimal control,” *ArXiv*, 2020.

[75] E. Mikuláš, M. Gulan, and G. Takács, “Model predictive torque vectoring control for a formula student electric racing car,” in *Proceedings 17th European Control Conference*, 2018, pp. 581–588.

[76] R. E. Ritschel, F. Schrödel, J. Hädrich, and J. Jäkel, “Nonlinear model predictive path-following control for highly automated driving,” *IFAC PapersOnLine*, 2019.

[77] M. Castillo-Lopez, P. Ludivig, S. A. Sajadi-Alamdari, J. L. Sanchez-Lopez, M. A. Olivares-Mendez, and H. Voos, “A real-time approach for chance-constrained motion planning with dynamic obstacles,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3620–3625, 2020.

[78] M. Diehl and S. Gros, *Numerical Optimal Control (Draft)*.

[79] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, “qpOASES: A parametric active-set algorithm for quadratic programming,” *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[80] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modeling, Planning and Control*, 3rd ed. Springer, 2008.

[81] S. Haddadin, A. De Luca, and A. Albu-Schäffer, “Robot collisions: A survey on detection, isolation, and identification,” *IEEE Transaction on Robotics*, vol. 33, no. 6, pp. 1292–1312, 2017.

[82] E. Magrini and A. De Luca, “Hybrid force/velocity control for physical human-robot collaboration tasks,” in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016.

[83] J. Hollerbach, W. Khalil, and M. Gautier, “Model identification,” in *Handbook of Robotics*, 2nd ed. Springer, 2016, pp. 113–138.

[84] A. Janot, P. Vandanjon, and M. Gautier, “A generic instrumental variable approach for industrial robot identification,” *IEEE Transactions on Control Systems Technology*, vol. 22, no. 1, pp. 132–145, 2014.

[85] C. Gaz, F. Flacco, and A. De Luca, “Extracting feasible robot parameters from dynamic coefficients using nonlinear optimization methods,” in *Proceedings IEEE International Conference on Robotics and Automation*, 2016, pp. 2075–2081.

[86] W. Khalil and E. Dombre, *Modeling, Identification and Control of Robots*. Hermes Penton London, 2002.

[87] C. Gaz and A. De Luca, “Payload estimation based on identified coefficients of robot dynamics — with an application to collision detection,” in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2017, pp. 3033–3040.

[88] M. Deisenroth, D. Fox, and C. Edward Rasmussen, “Gaussian processes for data-efficient learning in robotics and control,” *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 37, pp. 408–423, 02 2015.

[89] S. Kamthe and M. Deisenroth, “Data-efficient reinforcement learning with probabilistic model predictive control,” in *Proceedings International Conference on Artificial Intelligence and Statistics*, 2018.

[90] F. Berkenkamp and A. P. Schoellig, “Safe and robust learning control with gaussian processes,” in *Proceedings 14th European Control Conference*, July 2015, pp. 2496–2501.

[91] C. Ostafew, A. Schoellig, and T. D. Barfoot, “Robust constrained learning-based NMPC enabling reliable mobile robot path tracking,” *The International Journal of Robotics Research*, vol. 35, 05 2016.

[92] Z. Shareef, P. Mohammadi, and J. Steil, “Improving the inverse dynamics model of the KUKA LWR IV+ using independent joint learning,” *IFAC PapersOnLine*, vol. 49, no. 21, 2016.

[93] T. Waegeman, F. Wyffels, and B. Schrauwen, “Feedback control by online learning an inverse model,” *IEEE Transaction on Neural Networks and Learning Systems*, vol. 23, pp. 1637–1648, 10 2012.

[94] J. Umlauft, T. Beckers, M. Kimmel, and S. Hirche, “Feedback linearization using gaussian processes,” in *Proceedings IEEE 56th Conference on Decision and Control*, 2017, pp. 5249–5255.

[95] D. Nguyen-Tuong, M. Seeger, and J. Peters, “Computed torque control with nonparametric regression models,” in *Proceedings American Control Conference*, 2008, pp. 212–217.

[96] C. Gaz, F. Flacco, and A. De Luca, “Identifying the dynamic model used by the KUKA LWR: A reverse engineering approach,” *Proceedings IEEE International Conference on Robotics and Automation*, pp. 1386–1392, 09 2014.

[97] F. Al-Bender, V. Lampaert, and J. Swevers, “Modeling of dry sliding friction dynamics: From heuristic models to physically motivated models and back,” *Chaos*, vol. 14, pp. 446–60, 07 2004.

[98] A. De Luca and W. Book, “Robots with flexible elements,” in *Handbook of Robotics*, 2nd ed. Springer, 2016, pp. 243–282.

[99] G. Oriolo and Y. Nakamura, “Control of mechanical systems with second-order nonholonomic constraints: Underactuated manipulators,” in *Proceedings 30th IEEE Conference on Decision and Control*, 1991, pp. 2398–2403.

[100] A. De Luca, S. Iannitti, R. Mattone, and G. Oriolo, “Underactuated manipulators: Control properties and techniques,” *Machine Intelligence and Robotic Control*, vol. 4, no. 3, pp. 113–125, 2002.

[101] M. Spong, "Partial feedback linearization of underactuated mechanical systems," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1994, pp. 314–321.

[102] ——, "Underactuated mechanical systems," in *Control Problems in Robotics and Automation*, B. Siciliano and K. Valavanis, Eds. Springer, 1998, pp. 135–150.

[103] I. Fantoni and R. Lozano, "The Pendubot system," in *Non-linear Control for Underactuated Mechanical Systems*. Springer, 2002, pp. 53–72.

[104] O. Kolesnichenko and A. Shiriaev, "Partial stabilization of underactuated Euler–Lagrange systems via a class of feedback transformations," *IEEE Control Systems Letters*, vol. 45, no. 2, pp. 121–132, 2002.

[105] Y. Orlov, L. T. Aguilar, L. Acho, and A. Ortiz, "Swing up and balancing control of Pendubot via model orbit stabilization: Algorithm synthesis and experimental verification," in *Proceedings 45th IEEE Conference on Decision and Control*, 2006, pp. 6138–6143.

[106] T. Albahkali, R. Mukherjee, and T. Das, "Swing-up control of the Pendubot: An impulse–momentum approach," *IEEE Transaction on Robotics*, vol. 25, no. 4, pp. 975–982, 2009.

[107] A. R. Ansari and T. D. Murphey, "Sequential action control: Closed-form optimal control for nonlinear and nonsmooth systems," *IEEE Transaction on Robotics*, vol. 32, no. 5, pp. 1196–1214, 2016.

[108] D. Qian, J. Yi, and D. Zhao, "Hierarchical sliding mode control to swing up a Pendubot," in *Proceedings American Control Conference*, 2007, pp. 5254–5259.

[109] S. Moussaoui, A. Boulkroune, and S. Vaidyanathan, "Fuzzy adaptive sliding-mode control scheme for uncertain underactuated systems," in *Advances and Applications in Nonlinear Control Systems*, S. Vaidyanathan and C. Volos, Eds. Springer, 2016, pp. 351–367.

[110] M. P. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings 28th International Conference on Machine Learning*, 2011, pp. 465–472.

[111] K. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, and J.-B. Mouret, "Black-box data-efficient policy search for robotics," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017, pp. 51–58.

[112] M. Saveriano, Y. Yin, P. Falco, and D. Lee, "Data-efficient control policy search using residual dynamics learning," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017, pp. 4709–4715.

[113] A. Schoellig and R. D'Andrea, "Optimization-based iterative learning control for trajectory tracking," in *Proceedings 10th European Control Conference*, 2009, pp. 1505–1510.

[114] F. L. Mueller, A. Schoellig, and R. D'Andrea, "Iterative learning of feed-forward corrections for high-performance tracking," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 3276–3281.

[115] S. Gillen, M. Molnar, and K. Byl, "Combining deep reinforcement learning and local control for the Acrobot swing-up and balance task," in *Proceedings 59th IEEE Conference on Decision and Control*, 2020, pp. 4129–4134.

[116] P. Zhang, X. Lai, Y. Wang, and M. Wu, "Motion planning and adaptive neural sliding mode tracking control for positioning of uncertain planar underactuated manipulator," *Neurocomputing*, vol. 334, pp. 197–205, 2019.

[117] K. Chen, J. Yi, and D. Song, "Gaussian processes model-based control of underactuated balance robots," in *Proceedings International Conference on Robotics and Automation*, 2019, pp. 4458–4464.

[118] M. Capotondi, G. Turrisi, C. Gaz, V. Modugno, G. Oriolo, and A. De Luca, "Learning feedback linearization control without torque measurements," in *Proceedings 2nd Italian Conference on Robotics and Intelligent Machines*, 2020, pp. 131–134.

[119] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv*, 2017.

[120] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *Proceedings 34th International Conference on Machine Learning*, vol. 70, 2017, pp. 22–31.

[121] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3387–3395.

[122] S. Heim, A. Rohr, S. Trimpe, and A. Badri-Spröwitz, "A learnable safety measure," in *Proceedings Machine Learning Research (3rd Conference on Robot Learning)*, 2019.

[123] M. Zhong, M. Johnson, Y. Tassa, T. Erez, and E. Todorov, "Value function approximation and model predictive control," in *Proceedings IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2013, pp. 100–107.

[124] A. Tamar, G. Thomas, T. Zhang, S. Levine, and P. Abbeel, "Learning from the hindsight plan - episodic MPC improvement," in *Proceedings IEEE International Conference on Robotics and Automation*, 2017, pp. 336–343.

[125] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *Proceedings IEEE International Conference on Robotics and Automation*, 2018, pp. 7559–7566.

[126] S. Kamthe and M. P. Deisenroth, "Data-efficient reinforcement learning with probabilistic model predictive control," in *Proceedings 21st International Conference on Artificial Intelligence and Statistics*, 2018, pp. 1701–1710.

[127] N. Mansard, A. Del Prete, M. Geisert, S. Tonneau, and O. Stasse, "Using a memory of motion to efficiently warm-start a nonlinear predictive controller," in *Proceedings IEEE International Conference on Robotics and Automation*, 2018, pp. 2986–2993.

[128] F. Farshidian, D. Hoeller, and M. Hutter, "Deep value model predictive control," in *Proceedings Machine Learning Research (3rd Conference on Robot Learning)*, 2019.

[129] M. Zanon and S. Gros, "Safe reinforcement learning using robust mpc," *IEEE Transactions on Automatic Control*, vol. 66, no. 8, pp. 3638–3652, 2021.

[130] K. P. Wabersich, L. Hewing, A. Carron, and M. N. Zeilinger, "Probabilistic model predictive safety certification for learning-based control," *IEEE Transactions on Automatic Control*, vol. 67, no. 1, pp. 176–188, 2022.

[131] V. Makoviychuk, L. Wawrzyniak, Y. Guo, K. S. M. Lu, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu based physics simulation for robot learning," in *Proceedings Conference on Neural Information Processing Systems*, 2021.

[132] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, "Brax - a differentiable physics engine for large scale rigid body simulation," *ArXiv*, 2021.

[133] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," *ArXiv*, 2021.