SAPIENZA
UNIVERSITÀ DI ROMA

Model learning for trajectory tracking of robot manipulators

Sapienza University of Rome

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

Candidate

Marco Capotondi

ID number 1454279

Thesis Advisor

Prof. Alessandro De Luca

January 2023

Thesis defended on January 2023
in front of a Board of Examiners composed by:

Prof.ssa Paola CAPPANERA, Università di Firenze

Prof. Danilo PANI, Università di Cagliari

Prof. Paolo VALIGI, Università di Perugia

Model learning for trajectory tracking of robot manipulators
Ph.D. thesis. Sapienza – University of Rome

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

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

Abstract

Model based controllers have drastically improved robot performance, increasing task accuracy while reducing control effort. Nevertheless, all this was realized with a very strong assumption: the exact knowledge of the physical properties of both the robot and the environment that surrounds it. This assertion is often misleading: in fact modern robots are modeled in a very approximate way and, more important, the environment is almost never static and completely known. Also for systems very simple, such as robot manipulators, these assumptions are still too strong and must be relaxed. Many methods were developed which, exploiting previous experiences, are able to refine the nominal model: from classic identification techniques to more modern machine learning based approaches. Indeed, the topic of this thesis is the investigation of these data driven techniques in the context of robot control for trajectory tracking. In the first two chapters, preliminary knowledge is provided on both model based controllers, used in robotics to assure precise trajectory tracking, and model learning techniques. In the following three chapters, are presented the novelties introduced by the author in this context with respect to the state of the art: three works with the same premise (an inaccurate system modeling), an identical goal (accurate trajectory tracking control) but with small differences according to the specific platform of application (fully actuated, underactuated, redundant robots). In all the considered architectures, an online learning scheme has been introduced to correct the nominal feedback linearization control law. Indeed, the method has been primarily introduced in the literature to cope with fully actuated systems, showing its efficacy in the accurate tracking of joint space trajectories also with an inaccurate dynamic model. The main novelty of the technique was the use of only kinematics information, instead of torque measurements (in general very noisy), to online retrieve and compensate the dynamic mismatches. After that the method has been extended to underactuated robots. This new architecture was composed by an online learning correction of the controller, acting on the actuated part of the system (the nominal partial feedback linearization), and an offline planning phase, required to realize a dynamically feasible trajectory also for the zero dynamics of the system. The scheme was iterative: after each trial, according to the collected information, both the phases were improved and then repeated until the task achievement. Also in this case the method showed its capability, both in numerical simulations and on real experiments on a robotics platform. Eventually the method has been applied to redundant systems: differently from before, in this context the task consisted in the accurate tracking of a Cartesian end effector trajectory. In principle very similar to the fully actuated case, the presence of redundancy slowed down drastically the learning machinery convergence, worsening the performance. In order to cope with this, a redundancy resolution was proposed that, exploiting an approximation of the learning algorithm (Gaussian process regression), allowed to locally maximize the information and so select the most convenient self motion for the system; moreover, all of this was realized with just the resolution of a quadratic programming problem. Also in this case the method showed its performance, realizing an accurate online tracking while reducing both the control effort and the joints velocity, obtaining so a natural behaviour. The thesis concludes with summary considerations on the

proposed approach and with possible future directions of research.

Ringraziamenti

È viaggiando che si trova la saggezza.

Contents

1 Introduction **1**

2 Background on model-based robot trajectory control **7**
2.1 Computed torque 9
2.1.1 Inverse dynamics 9
2.1.2 Feedback linearization 9
2.1.3 Partial feedback linearization 11
2.1.4 Feedback linearization in task space and redundancy 12
2.2 Model predictive control 15

3 Background on model learning for robot control **19**
3.1 Model learning and identification 19
3.2 Linear parametrization of robot dynamics 20
3.2.1 Identification 21
3.2.2 Identification from motor currents with unknown signs 23
3.2.3 Adaptive control 24
3.3 Machine learning approaches 25
3.3.1 Model learning in robotics 25
3.3.2 The regression problem 28
3.3.3 Gaussian process regression 31
3.3.4 Approximate Gaussian process regression 40

4 Online learning with feedback linearization control of robots **45**
4.1 Motivation and contribution 45
4.2 Problem formulation 46
4.3 The proposed approach 46
4.3.1 Control architecture 47
4.3.2 Dataset collection procedure 49
4.3.3 Controllability Gramian 50
4.4 Simulation results 51
4.5 Chapter summary 52

5 Online learning for planning and control of underactuated robots **57**
5.1 Motivation and contribution 57
5.2 Problem formulation 59
5.3 The proposed iterative approach 60
5.3.1 Planning 61

5.3.2 Control 62
5.3.3 Dataset collection procedure for the active DOFs 63
5.3.4 Dataset collection procedure for the passive DOFs 63
5.4 Results 63
5.4.1 Simulation results 65
5.4.2 Experimental results 67
5.5 Chapter summary 70

6 Online learning and optimization for redundant robots 73
6.1 Motivation and contribution 73
6.2 Problem formulation 74
6.3 The proposed approach 75
6.3.1 Dataset collection procedure 76
6.3.2 Control architecture 76
6.3.3 Redundancy resolution and linGP 77
6.4 Simulation results 82
6.5 Chapter summary 83

7 Conclusions 87
7.1 Future outlook 89

Bibliography 91

Chapter 1

Introduction

The third industrial revolution, started in early 70's with manufacturing automation, has largely adopted robot manipulators technology. We can define a robot manipulator as a sequence of rigid bodies, called links, connected through actuated joints and which final link corresponds to the so called end effector, used to manipulate (from that *robot manipulator*) the environment and then realize a task. Since the design of these robot has remained basically the same over time, also the algorithms used for controlling these robots remained quite unchanged, so very simple and requiring a static and controlled environment. This was considered in general a valid assumption, at least for factories, since robots were confined in a cage separated from the operators. However technology and state of the art improvements allowed to refine these algorithms, considering the robot's entire structure and so move from simpler decentralized controllers (such as PID) to the so called model based approaches. The use of these controllers allowed to assure certain properties of the closed loop system, such as stability and convergence, to introduce a degree of optimality and feasibility in the controller's performance and, very important, to manage the robot interaction with the environment [1]. This last point is very important, since the use of model based controllers has revolutionized the human-robot interaction, allowing the coexistence and cooperation between the human and the machine in a shared environment, something unthinkable in 90's [2]. These convenient features are very desirable and influent in the robot's manipulator performance, but are be obtained at the cost of a precise knowledge of the considered system, something in general not possible. In particular for industrial hardware this is quite unrealistic: datasheets provided by robot's manufacturer are in fact very inadequate in providing information around the dynamic model of the robot (since not necessary for the final user). To obtain a more accurate model, identification procedures are usually realized that, through the reproduction of a specific motion for the various link of the robots, allows to find the optimal combination of the robot physical parameters that better matches with the torque measurements collected during these procedures. These methods are quite effective but must be performed offline and under certain conditions, such as fully actuation, presence of torque sensors and specific robot's structure that allows the linear parametrization of the system. On the other hand, if the goal is shifted from the robot identification to just the accurate tracking of a specific trajectory, then some online algorithms can be used,

Figure 1.1. The development of more sophisticated control techniques allowed the cooperation between human operator and robot in a shared environment. This was unthinkable in 90's, in which robot manipulators had to act in rigid environment (actually cages), with a predefined spatial and time separation with respect to humans. In this context, the maturation of model based controllers was fundamental. image: © KUKA Systems GmbH

such as adaptive controllers, that tries to find inline the best parameters combination for that specific motion or, as in the case of robust controllers, that considers the nonlinear effects generated by the use of a model based controller with inaccurate parameters as an external disturbance to compensate. These online approaches have guaranteed convergence properties, but also some drawbacks. In the first case the dynamics of these parameters will converge to a model that, unless the motion has specific features (the so called *exciting trajectories*), is not general, requiring the restart of the algorithm for a different task. Robust controllers instead assures in general accurate trajectory tracking at the cost of higher input effort, rough transient and the validity of certain assumptions (such as the disturbances bounded and matched with the input). Lastly, all the requirements of these methods may be so stringent that their actual implementation on real environment, so uncertain and not completely structured, can be impossible or with very poor results. The rebirth of machine learning techniques from 2000's influenced all science's fields, including control theory, providing a new way to account this old problem. The main idea of these new approaches is the use of past experiences as basin of knowledge from which the algorithm can learn and improve its performance. According to the definition of performance (loss, cost or reward, depending on the theoretical framework in which the algorithm is framed) and how these experience are generated, different kind of learning methods are suited. If the goal consists in the construction of the approximation of the mathematical model that relates a set of data, conventionally called input and output, the problem is defined as Supervised Learning. If it is not known the casual relation (so who is the input and the output) and the task is instead the reconstruction of the patterns that relates the entire set of data, the

problem is defined as Unsupervised Learning. Eventually, if these experiences are generated by the choice of an autonomous agent to optimize a specific performance, called cumulative reward, is called Reinforcement Learning. From these very generic families of algorithms are derived various methods, such as online learning, continual learning, transfer learning, curriculum learning and so on, each one suited for a particular task. The fact that these techniques are independent with respect to the application context and very adaptable enabled the use of data driven techniques in very different fields, from image processing to system control.

Given that, the control engineering community have recognised data driven approaches as one possible bridge between the classic control theory and its real world application. This is truly evident in robot control, in which the tasks complexity, the inevitable presence of uncertainties and the human-robot interaction require controllers able to adapt and cope with the so called *reality gap*.

The merging of robotics and machine learning, two distant (but not too much) fields, generated so a new and fascinating subject of research called robot learning, a topic very large containing different aspects such as perception, autonomous decision making, high and low level control and so on.

In this large spectre of arguments a particular one, called learning based control, is the one interesting for this work. It consists in the enrichment of the robot controller with previous experiences, to provide the missing knowledge and so improving the performance. Usually this is realized in two ways: model learning, consisting in the refinement of the controller's representation of the robot (the model), from which the control law is generated and policy learning, in which instead the control law (the policy) itself is updated. According to the specific task, one approach is more tailored than another. In presence of simple goal, model learning is convenient, since accurate reconstruction may allow to achieve a more optimal performance (w.r.t. tracking precision, control effort and constraints satisfaction) and re-usability on different tasks. In case of very complex goal, in which also accurate modeling doesn't assure the task satisfaction and the controller synthesis is very hard, is instead a common practice to introduce policy learning techniques, more keen on the goal realization and on providing an expertise to the controller for the specific task.

A classical example of policy Learning is the set of techniques that can be assimilated to Reinforcement Learning (RL). As mentioned before, RL is an approach based on the idea that, to maximize the performance and consequently solve a problem, possible solutions are generated according to the choice of an autonomous agent. The outcome of these choices, called actions, are then collected and used as learning experience for the agent's improvement. Reinforcement Learning has been applied in robotics, with some success [4], since its large potentiality in solving with an unique algorithmic structure a great variety of complex problems, from grasping [5, 6] to legged locomotion [7, 8, 9]. Despite the fact that is considered one of the most promising research field, this approach presents some issues that in general need to be accounted, such as the need of huge amount of experiences to explore the input space, the impossibility of qualifying and explaining the behaviour of the final controller in the different contexts and the effective transfer of these policies to real systems (*Sim-To-Real* problem) [3].

Model Learning approaches instead consist in the retrieving of an approximation of the system dynamics that can be used for the controller synthesis. Differently from

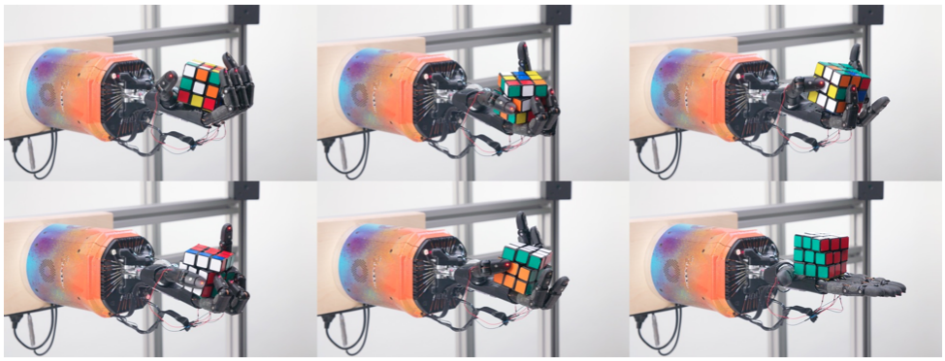

Figure 1.2. An example of a policy learning task: the resolution of the Rubik cube [3]. The modeling of the problem is quite simple and in general accurate, so the learning here is suited not to correct the nominal model but instead to understand the sequence of action required to obtain the specific goal. image: © OpenAI

continuous control theory, usually in robot learning the time evolution is considered discrete and integrated, so connecting the current pair state-input to the next state. According to what is the unknown part of this mapping, are defined the following macro categories [10]

- Forward model learning, consisting in the approximation of the mapping between the actual state-input and the next state
- Inverse model learning, consisting in the reconstruction of the function that provides the control input required to steer the system from the actual state to the next state
- Mixed model learning, in which forward and inverse model learning are jointly used in order to add consistency to the reconstruction
- Operator modeling learning, consisting in the reconstruction of the system's time evolution operator (useful for multi steps propagation of the system)

As just mentioned, model learning doesn't provide directly the requested control law (differently from policy learning). Instead, once that the model of the system has been retrieved (at least partially), classic model based controllers can incorporate this knowledge, to improve the nominal performance and enforcing some particular requirements/constraints that otherwise would have been unachievable. Many examples of this kind of approach will be presented in the manuscript.

In the present thesis is accounted the general problem of accurate trajectory tracking control for different robotics systems via online model learning algorithms. At first, in order to provide the correct preliminary knowledge to the reader, two brief summaries around model based controllers and model learning algorithms used in robotics are provided. Then the research results, obtained by the author on different platforms, are presented. Eventually, a conclusive analysis on the strengths, weakness and future improvement of these methods is performed.

This thesis is organized into seven chapters, shortly described as follows:

Figure 1.3. An example of model learning: the learning of the dynamics of a moving robot in an unknown terrain [11]. This technique is very suited for the task. The model of the system is in fact the unicycle, a very simple and unrealistic dynamics; the neglected remaining terms, that includes low level controller dynamics, slipping effects cause by the mud (in this particular case) and other disturbances is included through the learning. Eventually, on top of this corrected model, can be included a model based controller that exploits this knowledge (such as Model Predictive Controller in the paper). image: extracted from the paper [11]

- **Chapter 2 - Background on model-based robot trajectory control** This chapter provides a review of two different class of model-based controllers, *Computed Torque* and *Model Predictive Control*. The different approaches are presented, analyzing the sensitivity of these algorithms with respect to model uncertainties and degree of actuation.

- **Chapter 3 - Background on model learning for robot control** This chapter provides a review of various learning-based approaches for robotics, from classical methods such as linear parametrization and adaptive control to data driven ones. In particular Gaussian process regression technique is analyzed, considering its strengths, weaknesses, the range of application and how approximate approaches can mitigate the limitations of the method.

- **Chapter 4 - Online learning with feedback linearization control of robots** Inaccurate robot modeling generates unexpected dynamic effects, in particular when model-based controllers are used to command the system. The uncertainties in dynamical parameters, such as mass, inertia, friction, and unknown payloads, in fact can cause unexpected controller's behaviours, reducing the overall performance. In this chapter is presented, a learning-based version of a feedback-linearization controller that is able to compensate these effects online, improving the tracking performance in just few algorithms steps

and realizing all of this without using torque sensors. The proposed approach has been tested on a simulated KUKA LWR4+ robot and presented during the Conference of Robotics Learning (CoRl) in 2019 [12].

- **Chapter 5 - Online learning for planning and control of underactuated robots** A common approach for realizing specific task with underactuated robots consists in a preliminary offline planning phase, performed to generate a dynamically aware trajectory, and then an online phase in which the proposed trajectory is tracked trough the use of model based controllers. In this case, the problem of uncertainties in the dynamic model not only influences the controller performance, as for fully actuated systems, but even the planning phase, providing an inaccurate transition model that generates a not feasible trajectory, which accurate tracking for definition is impossible. In this chapter is proposed an iterative approach, composed of sequences of planning and control phases, in order to manage the aforementioned problems. The proposed approach has been tested experimentally on a two-link underactuated robot, the Pendubot, and it was published in the IEEE Robotics and Automation Letters (RA-L) in 2022 [13].

- **Chapter 6 - Online learning and optimization for redundant robots** Accurate trajectory tracking in the task space is critical in many robotics applications. Model-based robot controllers are able to ensure very good tracking but lose effectiveness in the presence of model uncertainties. On the other hand, online learning-based control laws can handle poor dynamic modeling, as long as prediction errors are kept small and decrease over time. However, in the case of redundant robots directly controlled in the task space, this condition is not usually met. In this chapter is presented an online learning-based control fashion that exploits the robot redundancy so as to increase the overall performance and shorten the learning transient. The validity of the proposed approach is shown through a comparative study conducted on a simulated KUKA LWR4+ robot and presented in the 4-th Conferenza Italiana di Robotica e Macchine Intelligenti (IRIM) in 2022 [14].

- **Chapter 7 – Conclusions** In this chapter a final discussion on the contributions and on the future improvements of the proposed methods is presented.

Chapter 2

Background on model-based robot trajectory control

Together with the technological evolution of the mechanical automated system called robot during third industrial revolution, a coherent theoretical architecture was born and developed to better characterize these systems and so improve mechanical design and controllers synthesis. In particular, since the limited hardware and computation power (nowadays limitations almost disappeared), the modeling of these systems were kept simple but still representative of the dominant dynamics. A classical approach for deriving motion equations of rigid serial robot manipulator consists in considering the system as a chain of rigid bodies, called links, connected each other through joints. These joints are kinematic constraints that impose to the subsequent link a particular relative motion, that can be a rotation around the joint axis (rotating joints) or translation along the joint axis (prismatic joints). These kinematic limitations are associated conventionally to the fact that the mechanical actuation of this class of system overlaps the joints structure, so providing only compatible motions. Since the system is eventually composed by a set of rigid bodies and kinematic constraints acting on them, Lagrange formulation of the dynamic is very convenient. In this framework in fact, independently from the reference frame, with a suitable change of coordinates the system is not represented anymore as the union of all kinematic coordinates of multiple rigid bodies but instead through the use of the so called *generalized coordinates*. As usual in Lagrangian mechanics, the dimension of the generalized coordinates vector n represents the actual degree of freedom of the system and is usually smaller than the number of coordinates that represents the same system in the Newtonian formulation. Formally speaking, it's possible to define the Lagrangian of the system as

$$L = T - U \tag{2.1}$$

in which T,U and L are respectively the Kinetic Energy, the Potential Energy and the Lagrangian, all of them dependent on the generalized coordinates $\boldsymbol{q} \in \boldsymbol{C}$, with $\boldsymbol{C}$ the space in which are defined called *Configuration Space*. With the application of variational principles are obtained the so called *Lagrange's Equations*, representing

the motion equations of the system, as follows

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\boldsymbol{q}}}\right)^T - \left(\frac{\partial L}{\partial \boldsymbol{q}}\right)^T = \boldsymbol{\tau} \tag{2.2}$$

in which $\boldsymbol{\tau}$ represents the generalized force, resulting from the satisfaction of the kinematic constraints not only acting on the rigid bodies coordinates but also on the external forces exerted on the system. According to the physical intuition, for serial rotating robot manipulators (such as KUKA LW4+) with open chain (so joints constraining kinematically only subsequent links), the q_i generalized coordinate represents the relative angle between the rotation axis of joint i and i+1. From further considerations on rigid bodies kinematics and dynamics [15], it is possible to represent both kinematic and potential energies of the system as nonlinear functions, depending respectively on $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and $\boldsymbol{q}$

$$\begin{aligned} T &= \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}} \\ U &= \sum_{i=1}^{N} -m_i \boldsymbol{g}_0^T \boldsymbol{r}_{0,CM_i}(\boldsymbol{q}) \end{aligned} \tag{2.3}$$

with $\boldsymbol{M}(\boldsymbol{q})$ representing the inertia matrix (so symmetric, positive definite and dependent only from the robot configuration and not from its derivative), m_i the mass of the i-th link, $\boldsymbol{g}_0$ the gravity vector and $\boldsymbol{r}_{0,ci}(\boldsymbol{q})$ the vector representing the position of the center of mass of the i-th link w.r.t. the reference frame 0 (the base frame). According to eq. 2.2, the motion equations are derived as:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} \tag{2.4}$$

with $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ the sum of Coriolis-Centrifugal terms , $\boldsymbol{g}(\boldsymbol{q})$ the Gravity component and $\boldsymbol{\tau}$ the generalized forces acting on the system

$$\begin{aligned} \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) &= \dot{\boldsymbol{q}}^T \boldsymbol{C}(\boldsymbol{q})\dot{\boldsymbol{q}} \\ \boldsymbol{g}(\boldsymbol{q}) &= \frac{\partial U}{\partial \boldsymbol{q}} \end{aligned} \tag{2.5}$$

In this representation the generalized forces acting on the systems represents a torque, which effect is the accelerations of the angular displacements between subsequent links (the generalized coordinates in case of rotating robots). They can be conventionally commanded as a control input (assigned via motors aligned with the joints axis) or considered as external disturbances deriving from the physical interaction of the robot with the environment (e.g static interaction of the end effector). These steps are the skeleton from which different model based controllers are designed. A classic task for these systems corresponds to the so called *Trajectory Tracking*. This goal correspond to assure that, assigned a specific reference trajectory (that can be also a single configuration, in the case of *Regulation* problem) the robot state (or a specific subset of them) will follow the same profile in time of the reference one. Considering eq. 2.4 is quite evident that robots manipulator are second order system, so having a relative degree $r = 2$. Given that is natural and physically consistent that, for both trajectory tracking and regulation, the reference trajectory is assumed at least to be

twice differentiable $\boldsymbol{q}_d \in C^2$ in the range of time $t \in [0, T]$ with T duration of the desired trajectory. Therefore, in order to not introduce any issue in the theoretical structure, continuity and differentiability will be always assured until the necessary degree. According to the specific task and available outputs, different controllers are then designed.

2.1 Computed torque

All the class of controllers that, given the nominal model of the robot, provides a torque feedback aware of the entire robot dynamics (and not only on the single joint dynamics, such as decentralized controller) are generally called *Computed Torque Controller*. In next section are presented, sorted according to complexity, classic computed torque controllers.

2.1.1 Inverse dynamics

Assuming that the initial condition of the system during the motion is the same as the reference one, the most basic approach is the providing of only a feedforward term $\boldsymbol{\tau}_d$

$$\boldsymbol{\tau} = \boldsymbol{\tau}_d = \boldsymbol{M}(\boldsymbol{q}_d)\ddot{\boldsymbol{q}}_d + \boldsymbol{c}(\boldsymbol{q}_d, \dot{\boldsymbol{q}}_d) + \boldsymbol{g}(\boldsymbol{q}_d) \tag{2.6}$$

This open loop controller, in case of nominal conditions and without external disturbances acting on the system, assures the exact reproduction of the desired motion. Since this ideal condition is never satisfied in real robots, an additive closed loop proportional-derivative term (PD) is introduced, in order to locally stabilize the error dynamics

$$\boldsymbol{\tau} = \boldsymbol{\tau}_d + \boldsymbol{K}_P(\boldsymbol{q}_d - \boldsymbol{q}) + \boldsymbol{K}_D(\dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}) \tag{2.7}$$

with $\boldsymbol{K}_P$ and $\boldsymbol{K}_D$ diagonal definite positive matrices. These gains are designed to shape the transient on the error dynamics and are quite dependent on the considered system and its digital implementation. A slightly more sophisticated controller, with respect to 2.7, is realized with the use of a variable PD that depends on the reference trajectory through the multiplicative term $\boldsymbol{M}(\boldsymbol{q}_d)$

$$\boldsymbol{\tau} = \boldsymbol{\tau}_d + \boldsymbol{M}(\boldsymbol{q}_d)\left(\boldsymbol{K}_P(\boldsymbol{q}_d - \boldsymbol{q}) + \boldsymbol{K}_D(\dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}})\right) \tag{2.8}$$

2.1.2 Feedback linearization

The previous controllers are quite simple (since linear) and can be easily implemented in real time systems, still assuring at least asymptotic stability (under certain conditions on the trajectory tracked and on the chosen gains). Despite of this, the performance is in general suboptimal and the behaviour of these systems during the convergence's transient can be very chaotic. More complex controllers were developed that, taking in account the entire robot model, are able to guarantee a better performance and a well shaped transient. In particular, a class of nonlinear controllers, called Feedback Linearization (FL), are easily implemented on rigid

robot manipulators. Roughly speaking, these controllers exploit the knowledge of the robot model to generate a nonlinear feedback which produces a linear and decoupled closed loop dynamics. These feedback linearized systems are then equivalent to a chain of integrators, which dimension correspond to the actual relative degree of the system. The complete expression of these controllers is grounded on the geometric formulation of nonlinear control theory, so in general requiring a large number of preliminary definitions and concepts. Nevertheless, in rigid robotics the explanation of these control laws is quite simple and physically grounded. Let's consider again the nominal model of the robot manipulator (2.4) and then a feedback term of the following form, with $\boldsymbol{u}$ control input still to be designed

$$\boldsymbol{\tau}_{FL} = \boldsymbol{M}(\boldsymbol{q})\boldsymbol{u} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) \tag{2.9}$$

After the application of the FL torque in eq. 2.4, with some algebric calculus is obtained the following closed loop dynamics

$$\ddot{\boldsymbol{q}} = \boldsymbol{u} \tag{2.10}$$

The resulting systems is now a double integrator (chain of $r = 2$ integrators), a linear and decoupled system which stability can easily assured through the shape of $\boldsymbol{u}$ as a proportional-derivative (PD) controller plus a feedforward term

$$\boldsymbol{u} = \ddot{\boldsymbol{q}}_d + \boldsymbol{K}_P(\boldsymbol{q}_d - \boldsymbol{q}) + \boldsymbol{K}_D(\dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}) \tag{2.11}$$

that, if is introduced a new state variable corresponding to the error $\boldsymbol{e} = \boldsymbol{q}_d - \boldsymbol{q}$, can be formalized as

$$\begin{aligned} &\ddot{\boldsymbol{q}} = \ddot{\boldsymbol{q}}_d + \boldsymbol{K}_P(\boldsymbol{q}_d - \boldsymbol{q}) + \boldsymbol{K}_D(\dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}) \rightarrow \\ &\ddot{\boldsymbol{e}} + \boldsymbol{K}_P\boldsymbol{e} + \boldsymbol{K}_D\dot{\boldsymbol{e}} = \boldsymbol{0} \end{aligned} \tag{2.12}$$

corresponding to a second order linear differential equation convergent to $\boldsymbol{0}$ with transient characteristics (such as rising time, elongation, settling time, bandwith and resonance frequency) depending on the gains $\boldsymbol{K}_P$ and $\boldsymbol{K}_D$ design. The steps here analyzed are quite simple and intuitive but still depends on a strong assumption: the accuracy of the nominal model used for the generation of the FL term. Let's characterize now the framework in a more realistic way, defining a nominal model (the one available for the controller's design) with same notation as before, apart the superscript on the matrices/vectors and the actual system model. The nominal model in general will be different from the real one and this difference can be represented as follows

$$\begin{aligned} \boldsymbol{M} &= \hat{\boldsymbol{M}} + \Delta\boldsymbol{M} \\ \boldsymbol{c} &= \hat{\boldsymbol{c}} + \Delta\boldsymbol{c} \\ \boldsymbol{g} &= \hat{\boldsymbol{g}} + \Delta\boldsymbol{g} \end{aligned} \tag{2.13}$$

The use of the same FL term as in 2.9 won't provide exact cancellation and the resulting closed loop dynamics will be different from the double integrator

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \hat{\boldsymbol{M}}(\boldsymbol{q})\boldsymbol{u} + \hat{\boldsymbol{c}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \hat{\boldsymbol{g}}(\boldsymbol{q}) \tag{2.14}$$

substitution of eq. 2.13 into eq. 2.14 will result eventually in

$$\begin{aligned} \ddot{\boldsymbol{q}} &= \boldsymbol{u} - \Delta \boldsymbol{M}\boldsymbol{u} - \Delta \boldsymbol{c} - \Delta \boldsymbol{g} \rightarrow \\ \ddot{\boldsymbol{q}} &= \boldsymbol{u} + \boldsymbol{\delta}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \end{aligned} \tag{2.15}$$

so a *perturbed* double integrator. Simple linear design of the control input $\boldsymbol{u}$ is now not enough for assuring convergence and stability of the system.

2.1.3 Partial feedback linearization

An important implicit assumption of the class of controllers explained in 2.1.2 is the fully actuation of the system. In general for an industrial robot manipulator this is a legit assumption, since their structure usually includes fully actuation on the joints or at least some kinematic constraints (such as for closed chain manipulators) that reduces the actual degrees of freedom of the robot. However, in order to account the same problem with this new class of system, a generalization of the previous framework has to be realized. Formally speaking, the eq. 2.4 must be modified in order to include the actuation matrix $\boldsymbol{A}(\boldsymbol{q})$, a state dependent matrix which geometric properties represent how the actuation input affects the generalized coordinates

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{A}(\boldsymbol{q})\boldsymbol{\tau} \tag{2.16}$$

where $\boldsymbol{q} \in \mathbb{R}^n$, $\boldsymbol{\tau} \in \mathbb{R}^m$, $\boldsymbol{A}(\boldsymbol{q}) \in \mathbb{R}^{n \times m}$ and $\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q})$ is the sum of Coriolis and Gravity term. When the actuation matrix $\boldsymbol{A}$ is full rank, through an input transformation and a change of coordinates the dynamics takes a partitioned structure [16] as follows

$$\begin{aligned} \boldsymbol{M}_{aa}(\boldsymbol{q})\ddot{\boldsymbol{q}}_a + \boldsymbol{M}_{ap}(\boldsymbol{q})\ddot{\boldsymbol{q}}_p + \boldsymbol{n}_a(\boldsymbol{q}, \dot{\boldsymbol{q}}) &= \boldsymbol{\tau} \\ \boldsymbol{M}_{pa}(\boldsymbol{q})\ddot{\boldsymbol{q}}_a + \boldsymbol{M}_{pp}(\boldsymbol{q})\ddot{\boldsymbol{q}}_p + \boldsymbol{n}_p(\boldsymbol{q}, \dot{\boldsymbol{q}}) &= \boldsymbol{0}, \end{aligned} \tag{2.17}$$

where $\boldsymbol{q} = \left(\boldsymbol{q}_a, \boldsymbol{q}_p\right)$ represents a partition of the DOFs, with $\boldsymbol{q}_a \in \mathbb{R}^m$ called active coordinates since collocated with the input $\boldsymbol{\tau}$ and the remaining $\boldsymbol{q}_p \in \mathbb{R}^{n-m}$ called passive coordinates. Also the inertia matrix $\boldsymbol{M}$ and the vector $\boldsymbol{n}$ of the remaining nonlinear terms are partitioned accordingly. From this very general formulation is properly defined the degree of underactuation $d = n - m$. This is very important, since allows to specify two different categories of robot:

- Fully Actuated, when $d = 0$
- Underactuated when $d > 0$

A procedure similar to 2.9 can be applied, called *Partial Feedback Linearization*. It consists in the definition of a suitable control input which effects is, in closed loop, to the linearization of one between the active/passive part of the dynamics and the introduction of a new nonlinear term in the remaining dynamics. Depending on whether the linearization is realized, on the active or passive component, is called

respectively *Collocated PFL* or *Not Collocated PFL* (collocation with respect to the input) which results eventually in

$$\begin{aligned} \ddot{\boldsymbol{q}}_{a,p} &= \boldsymbol{u} \\ \ddot{\boldsymbol{q}}_{p,a} &= \boldsymbol{h}(\boldsymbol{q}_a, \boldsymbol{q}_p, \dot{\boldsymbol{q}}_a, \dot{\boldsymbol{q}}_p, \boldsymbol{u}) \end{aligned} \tag{2.18}$$

The nonlinear part of the system is called in control theory as *Zero Dynamics*. This nomenclature is based on the fact that traditionally the goal of the linearized part is the steering of the state to the zero configuration (or to different configuration through a coordinate scaling). Usually the convergence to this equilibrium state is reached very fast (since the system is a double integrator, transients can be easily shaped, as said before), and the nonlinear remaining dynamics is analyzed when the equilibrium has been reached. For example, in the case of collocated linearization, assuming that $(\boldsymbol{q}_a, \dot{\boldsymbol{q}}_a, \boldsymbol{u}) = \boldsymbol{0}$

$$\ddot{\boldsymbol{q}}_p = \boldsymbol{h}(\boldsymbol{0}, \boldsymbol{q}_p, \boldsymbol{0}, \dot{\boldsymbol{q}}_p, \boldsymbol{0})$$

from this the term zero dynamics. Trajectory tracking task in this context can be easily realized on the linearized degree of freedom. Still, the zero dynamics is an autonomous system (not explicitly dependent from the control input) and, more important, acts as a dynamic constraint that limits widely the controllability of the entire robot. In fact may not exist a control input that satisfies the dynamic constraint and at the same time brings the system to an arbitrary state in a finite time. A way to keep in account all of these peculiarities is the introduction of a planning phase, such that the trajectory that the system has to track is compatible with both the active and passive coordinates [17]. In order to obtain this feasible motion, the planner should use the dynamic of the robot as transition model; this clearly points out why a rough nominal model can generate trajectories not really compatible with the real system. In summary, the linearized part has the same weakness highlighted in 2.14 (regarding the PFL); furthermore, an inaccurate nominal model generates a trajectory that could not be tracked also using an ideal controller.

2.1.4 Feedback linearization in task space and redundancy

Redundancy is a concept quite dependent on the actual task of the robot. Formally speaking, the task for a robot consists in a time varying constraints which satisfaction corresponds to the task execution. Usually, for robot manipulators, these constraints are dependent only to the robot configurations (the so called geometric constraints); classic examples of this are the Cartesian tasks, such as end effector position or orientation. Redundancy is generated by the fact that the task's dimension (so the number of scalar constraints acting on the system) is smaller than the dimension of the generalized coordinates vector. The task $\boldsymbol{h}(\boldsymbol{q}, \boldsymbol{x}) \in \mathbb{R}^p$ is defined as

$$\boldsymbol{h}(\boldsymbol{q}, \boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{0} \tag{2.19}$$

with $\boldsymbol{x} \in \mathbb{R}^p$ a value assigned for the specific task (in general time-dependent) and $\boldsymbol{q} \in \mathbb{R}^n$, a robot is redundant, with respect to the specific goal $\boldsymbol{h}$, if the degree of redundancy $k = n - p$ is larger than 0. Is important to note that in the case that

$n > p$ the mapping is not unique, since given a x there is an entire subspace of the Configuration Space which satisfies $\boldsymbol{h}(\boldsymbol{q}, \boldsymbol{x}) = \mathbf{0}$, (not only a single configuration). This is expressed in a more easily way saying that the mapping $f(q) : C \to \mathbb{R}^p$ is not injective and consequently invertible. These constraints usually belongs to the C^∞ functional space, so can be derived infinite times. Let's consider for now a degree of redundancy $k = 0$, so $p = n$ (so no redundancy). Time differentiation of $\boldsymbol{h}(\boldsymbol{q}, \boldsymbol{x})$ can be exploited to merge coherently the constraints with the robot dynamics. The time derivative of eq. 2.19 generates

$$\dot{\boldsymbol{h}}(\boldsymbol{q}, \boldsymbol{x}) = \dot{\boldsymbol{x}} - \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} = \mathbf{0} \tag{2.20}$$

with $\boldsymbol{J} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}(\boldsymbol{q})$ the Jacobian matrix, with $\boldsymbol{J} \in \mathbb{R}^{n \times p}$. A second time differentiation brings to

$$\ddot{\boldsymbol{h}}(\boldsymbol{q}, \boldsymbol{x}) = \ddot{\boldsymbol{x}} - \boldsymbol{J}(\boldsymbol{q})\ddot{\boldsymbol{q}} - \dot{\boldsymbol{J}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} = \mathbf{0} \tag{2.21}$$

Since the underlying fully actuated dynamics is 2.4, a FL approach can be exploited to shape the closed loop acceleration of the system (such that $\ddot{\boldsymbol{q}} = \boldsymbol{u}$). This modifies the previous equations

$$\ddot{\boldsymbol{x}} - \boldsymbol{J}(\boldsymbol{q})\boldsymbol{u} - \dot{\boldsymbol{J}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} = \mathbf{0} \tag{2.22}$$

with $\boldsymbol{u}$ additional control input that has to be designed. Differently from before, this new degree of freedom can be used to change the closed loop **task dynamics**, for example introducing a nonlinear canceling term and a feedforward+PD controller in the task space, such that

$$\boldsymbol{u} = \boldsymbol{J}^{-1}\left(\dot{\boldsymbol{J}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \ddot{\boldsymbol{x}}_d + \boldsymbol{K}_P(\boldsymbol{x}_d - \boldsymbol{x}) + \boldsymbol{K}_D(\dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}})\right) \tag{2.23}$$

the closed loop task dynamics is in the following form (considering $\boldsymbol{e}_x = \boldsymbol{x}_d - \boldsymbol{x}$)

$$\ddot{\boldsymbol{e}}_x + \boldsymbol{K}_P\boldsymbol{e}_x + \boldsymbol{K}_D\dot{\boldsymbol{e}}_x = \mathbf{0} \tag{2.24}$$

This can be realized since $\boldsymbol{J}$ is a quadratic matrix ($n = p$) and the inverse matrix $\boldsymbol{J}^{-1}$ is defined. Let's consider now the case of a redundancy degree $k > 0$, so $n > p$. Now the Jacobian matrix is a rectangular matrix and inversion is an operation not admissible. In fact, according to well known properties of matrices, the dimension of the null space associated with a rectangular matrix $\boldsymbol{J} \in \mathbb{R}^{nxk}$ is

$$\dim(\mathcal{N}(\boldsymbol{J})) \geq \min(k, n) \tag{2.25}$$

This inequality is in general true and, since the Jacobian is a configuration dependent mapping, becomes an equality only in the case of full ranked matrix (so when the configuration is not singular). In order to cope with a non square matrix, a generalization of the inversion was is defined in matrix theory, called *Pseudoinversion*. Given the matrix $\boldsymbol{J}$, the pseudoinverse matrix $\boldsymbol{J}^\#$ is the only matrix which satisfies the following properties:

$$\boldsymbol{J}\boldsymbol{J}^\#\boldsymbol{J} = \boldsymbol{J} \tag{2.26}$$

$$\boldsymbol{J}^\#\boldsymbol{J}\boldsymbol{J}^\# = \boldsymbol{J}^\# \tag{2.27}$$

$$(\boldsymbol{J}^\#\boldsymbol{J})^T = \boldsymbol{J}^\#\boldsymbol{J} \tag{2.28}$$

$$(\boldsymbol{J}\boldsymbol{J}^\#)^T = \boldsymbol{J}\boldsymbol{J}^\# \tag{2.29}$$

$$\boldsymbol{J}^\# = \boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1} \text{ when the matrix } \boldsymbol{J} \text{ is full rank} \tag{2.30}$$

There is another way to define pseudoinversion in a less axiomatic way. Let's consider the following linear system $\boldsymbol{A}(\boldsymbol{y}-\boldsymbol{y}_0)=\boldsymbol{b}$, with with $\boldsymbol{W}$ squared matrix. Since $\boldsymbol{A}$ is a non square matrix and there are (for redundant systems) more DOFs than equations, the system is undetermined; this means an infinite number of solutions. A way to select one explicit class of solutions, called *regularization*, is to find, within the all possible solutions, the one that minimizes its weighted norm. For realize that is formulated the following optimization problem

$$\begin{aligned}\boldsymbol{y}_{\text{pseudo},\boldsymbol{W}} = \min_{\boldsymbol{y}} \ & (\boldsymbol{y}-\boldsymbol{y}_0)^T\boldsymbol{W}^2(\boldsymbol{y}-\boldsymbol{y}_0) \\ \text{subject to } & \boldsymbol{A}\boldsymbol{y}-\boldsymbol{b}=\boldsymbol{0}\end{aligned} \tag{2.31}$$

It consists in a quadratic programming problem with linear constraints, that can be easily solved using Lagrange multipliers method. The analytical solution of the problem is expressed as follows

$$\boldsymbol{y}=\boldsymbol{y}_0+\boldsymbol{W}^{-1}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{W}^{-1}\boldsymbol{A}^T)^{-1}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{y}_0) \tag{2.32}$$

which can be rearranged, collecting common terms, as

$$\boldsymbol{y}=\boldsymbol{A}_{\boldsymbol{W}}^{\#}\boldsymbol{b}+\boldsymbol{P}_{\boldsymbol{W}}\boldsymbol{y}_0 \tag{2.33}$$

This represents the general solution of the linear system, in which the first term $\boldsymbol{A}_{\boldsymbol{W}}^{\#}\boldsymbol{b}$ is called the weighted pseudoinverse solution, the minimum weighted norm solution when $\boldsymbol{y}_0=\boldsymbol{0}$

$$\boldsymbol{A}_{\boldsymbol{W}}^{\#}\boldsymbol{b}=\boldsymbol{W}^{-1}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{W}^{-1}\boldsymbol{A}^T)^{-1}\boldsymbol{b} \tag{2.34}$$

and the second term is the projection in the null space of $\boldsymbol{A}$ (through the projector $\boldsymbol{P}_{\boldsymbol{W}}$) of the term $\boldsymbol{y}_0$

$$\boldsymbol{P}_{\boldsymbol{W}}\boldsymbol{y}_0=(\boldsymbol{I}-\boldsymbol{A}_{\boldsymbol{W}}^{\#}\boldsymbol{A})\boldsymbol{y}_0 \tag{2.35}$$

The projector matrix is characterized by the following property $\boldsymbol{A}\boldsymbol{P}_{\boldsymbol{W}}=\boldsymbol{0}$; this means that the solution is defined up to a constant value $\boldsymbol{P}_{\boldsymbol{W}}\boldsymbol{y}_0$, belonging to the null space of the matrix $\boldsymbol{A}$.

This definition can be clearly applied also for the case of the inverse kinematic, in view of eq. 2.20, substituting $\boldsymbol{A}=\boldsymbol{J}$, $\boldsymbol{y}=\dot{\boldsymbol{q}}$, $\boldsymbol{b}=\dot{\boldsymbol{x}}$ and $\boldsymbol{W}=\boldsymbol{I}$. Must be noticed that weighted pseudoinverse and pseudoinverse in general don't coincide; in particular the features listed in eq. 2.26 are partially satisfied by the weighted pseudoinverse. Thus in this context are equivalent, since the weight matrix coincides with the identity matrix. Independently from the specific characterization, eq. 2.23 must be modified, replacing the inverse matrix with the corresponding pseudoinverse, obtaining as follows

$$\boldsymbol{u}=\boldsymbol{J}^{\#}\left(\dot{\boldsymbol{J}}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}+\ddot{\boldsymbol{x}}_d+\boldsymbol{K}_P(\boldsymbol{x}_d-\boldsymbol{x})+\boldsymbol{K}_D(\dot{\boldsymbol{x}}_d-\dot{\boldsymbol{x}})\right) \tag{2.36}$$

Considering now the pseudoinversion of eq. 2.36, is evident that is always defined up to an acceleration term belonging to the null space of $\boldsymbol{J}$. According to this, can be designed a new control input $\boldsymbol{w}$ as $\boldsymbol{w}=\boldsymbol{u}+\ddot{\boldsymbol{q}}_{null}$, having the the same effect

on task dynamics of $\boldsymbol{u}$, but providing a different joints acceleration (and so joint's state evolution). This is very convenient, since allow sthe optimization of this null space acceleration (called *self-motion*) to execute another secondary task, such as joints reconfiguration (for obstacle avoidance) or minimization of the norm of the velocity/torque vector. This procedure, called *Redundancy Resolution*, can be realized analytically as a constrained optimization problem or locally through gradient based algorithms such as *Projected Gradient* [18]. Is very important the fact that this optimization can be realized only considering the actual robot configuration. The Jacobian represents in fact the tangent plane (a linear subspace) of the constraints in the specific configuration and any modification of the robot state may completely change the corresponding curvature and consequently the null space. As for previous approaches, this scheme is based on the assumption of accurate nominal dynamics: any dynamic mismatch generates an inaccurate feedback linearization, introducing a state dependent disturbance in both the task and joints closed loop dynamics, neither realizing the trajectory tasking nor the null space optimization.

2.2 Model predictive control

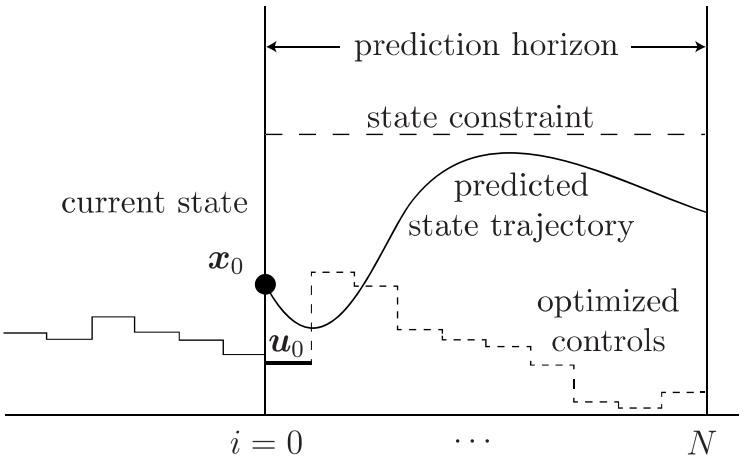

Figure 2.1. Scheme of the MPC strategy. At each control step, the optimization problem is solved starting from the current state $\boldsymbol{x}_0$ and only the first control inputs $\boldsymbol{u}_0^{\text{opt}}$ is applied to the system. Then the system is sampled again and the procedure reiterated. Image: © James B. Rawlings, David Q. Mayne, Moritz M. Diehl *Model Predictive Control: Theory, Computation, and Design*, 2022 Second Edition

A very different but still very valid class of model based controllers are the ones grounded in the Optimal Control theory. These techniques rely on a complete knowledge of the model and are applied both on liner and nonlinear systems. The main idea of these approaches is to resolve an optimization problem, considering the system dynamic as a constraint and imposing a specific loss function according to the tasks, such as integrated tracking error or control effort. These techniques are very effective, since constraints can be considered (such as torque saturation, velocity limits etc...), obtaining a more realistic policy. There exists many techniques originating from this scheme, such as the ones deriving from the exploitation of

Pontryagin maximum principle or optimal feedback controller as *Linear Quadratic Regulator* (LQR) [19]. Usually these controllers are designed offline, considering only the nominal condition of the system; however, the presence of unmodeled effects and external disturbances can reduce their actual effectiveness. What happens is in fact that the boundary conditions of the problem, such as the nominal evolution in time or exerted control effort, are completely different with respect to the real ones and, consequently, the corresponding optimal solution. To cope with this, were developed techniques that updates the optimization problem iteratively, so including the new measurements (and so the real conditions) at each step. The most known of these procedures is the so called *Model Predictive Controller*(MPC). Since the problem's conditions continuously update, is quite hard to provide a unique control law; indeed, the purpose of the method is then shifted to provide an input optimal at least in a certain temporal range, called *prediction horizon*: this is the large difference from infinite horizon approaches, such as LQR. The horizon of the MPC like approaches is limited and influences copiously the performance: short horizon reduces optimality and can influence recursive feasibility; large horizon is desirable but in general impractical in real time control. Here is presented the classical nonlinear formulation, conceptually equivalent to the linear one [20]:

$$\begin{aligned} \min_{\boldsymbol{u}_0,\dots,\boldsymbol{u}_{N-1}} \quad & \sum_{i=0}^{N-1} J_{\text{stage}}(\boldsymbol{x}_i, \boldsymbol{u}_i) + J_{\text{terminal}}(\boldsymbol{x}_N) && (2.37) \\ \text{subject to} \quad & \boldsymbol{x}_{i+1} - \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) = \boldsymbol{0} && i = 0\dots, N-1 \\ & \boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{u}_i) \leq \boldsymbol{0} \\ & \boldsymbol{h}(\boldsymbol{x}_i, \boldsymbol{u}_i) = \boldsymbol{0} \end{aligned}$$

in which

- $\boldsymbol{u}_0, \dots, \boldsymbol{u}_{N-1}$ is the set of decision variables, corresponding to the input from the time t_0 to the time t_{N-1}, so N (prediction horizon) steps.
- $J(\boldsymbol{x}_i, \boldsymbol{u}_i)$ is the loss function, composed by a stage cost $J_{\text{stage}}(\boldsymbol{x}_i, \boldsymbol{u}_i)$, added up on the N steps, depending from the input and the state and by a terminal cost $J_{\text{terminal}}(\boldsymbol{x}_N)$ that depends instead only from the final state. For trajectory tracking task the cost is usually defined as integrated (summed in discrete case) norm of the punctual position error and the integrated norm of the control input vector. In general can be complicated at will of the designer.
- $\boldsymbol{x}_{i+1} - \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) = \boldsymbol{0}$ is the integrated robot dynamics (since connecting quantities from step i to step i+1) also called *transition model*. Usually is obtained via classical discretization techniques from continuous model equations.
- $\boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{u}_i)$ and $\boldsymbol{h}(\boldsymbol{x}_i, \boldsymbol{u}_i)$ are the nonlinear constraints acting on the systems, such as torque and velocities saturation, joint limits and so on.

The algorithm consists in many steps. The first one is the gathering of the system measurements and the updating of the boundary conditions the optimization problem. After that, the problem is solved (if exists a feasible solution) and is found the optimal sequence of inputs in the horizon preview $\boldsymbol{U}^{\text{opt}}$. After that, in the control

phase only the first input $\boldsymbol{u}_0^{\text{opt}}$ is executed and finally, according to sampling and control period, the process is iterated. In this way the aforementioned method is quite equivalent to a closed loop controller: in fact, given the available outputs, such as measurements of the state of the system or more complicate functions of it at time t_k, (redefined as t_0 in the optimization problem), is provided the corresponding control input $\boldsymbol{u}_k$. The MPC is very general purpose method and can act in many ways: as a planner, considering all the sequence of optimal sequence of inputs and the corresponding states (instead of just the first one); as the primary low level feedback controller (in case is correctly specified the low level transition model) or also as a high level feedback controller. Despite these very convenient features, that made it as one of the most studied and exploited algorithm in almost all field of applied control engineering, there is a very strong weaknesses in this approach. In fact, the correctness of the model is not advisable, but mandatory. As a matter of fact, in case of a imprecise modeling, the drift between the actual dynamics and the simulated one is nonlinearly propagated, potentially disrupting the system's behaviour with respect to the nominal one (in particular when the prediction horizon is large). Many works were focused on introducing robustness with respect to the system uncertainties. A classic is the so called *Tube MPC*, in which is realized a virtual restriction of the constraints acting on the systems, in order to contain the drift and still satisfying the original constraints. It is a very elegant formulation, easily accountable for linear systems [21] (with some assumption on the shape of the disturbances) through the use of sets operation (such as Minkowki sum, erosion etc...); nevertheless, it presents an high computational load (complicating its online use for large DOFs system such as robot) and the theory is not yet completely developed in the case of nonlinear dynamics. Another state of the art method to introduce a degree of robustness, more suited for handling parametric uncertainties of the systems, is called *Stochastic MPC*. In this formulation the dynamics mismatched is connected to the presence of a probabilistic disturbance acting on the system (e.g. in the case of robot dynamics acting on the nominal parameters of the system). Standard approach include the probabilistic formulation of the MPC, using the so called *Chance Constraints*, so a probabilistic rearrangement of the constraints. Usually this optimization is realized only for linear systems and assumes independence and Gaussianity of the disturbance. A more simple way to account this issue is the so called *Scenario Based*: the algorithm samples different instances of the disturbances (according to its statistical distribution) and for each one of these solves a different optimization problem; after that, the final policy is obtained merging all the policy obtained individually, mixing them according to the statistical weights assigned to each instance [22]. This procedure is theoretically related to the chance constraints approach, approximating it; still it provides a very good performance and the same time reduces drastically the corresponding computational burden. All these methods (Robust and Stochastic) assume as starting point some knowledge around the disturbances influencing on the system, (such as magnitude bounds or the specific channels where the disturbance acts); in general this information is not granted. A more general way to account these problems, for all the class of controllers analyzed up to here, is the use of model learning techniques: these class of techniques usually doesn't require specific information around the missing part of the model and can be applied in all the model based controllers presented in this chapter.

Chapter 3

Background on model learning for robot control

As said before, robot learning is a very large field, composed by many subtopics. The subject that is taken in account in this thesis, the model learning for trajectory tracking control, belongs to the larger branch of learning for control. As mentioned before, these techniques consist in the improvement of controller through the use of previous experiences, either bettering the robot model or directly modifying the controller. The data blocks that, after a specific adjustment, provide the knowledge required for the application of the learning algorithms consists, in the robotics context, in sensors measurements such as images from camera, encoder measurements or accelerations from IMU. These quantities in fact allow the reconstruction of both the state of the robot and of the environment, the key facts for understanding the system and the effects of its actions.

In particular, from here on a brief overview on model learning technique is provided, specifying how the task is both arranged in robotics and in the machine learning framework, specifying in what consists and reviewing many methods state of the art for performing the considered objective. Final sections are then focused on a specific nonparametric technique, called *Gaussian process regression*, and its approximation.

3.1 Model learning and identification

The knowledge of a robot model is a preeminent fact in the study of its behaviour, both analytically or through numerical simulations. As shown in the previous chapter, for a specific class of systems (such as rigid robot manipulators), a mathematical description represented by motion equations can be found applying classical mechanics methods. This is formulation is in general valid for a very narrow category of robots and often very unreliable, since the presence of uncertainties and/or time varying dynamics (e.g shifting friction due to aging). Moreover, some systems such as soft robots cannot be modeled exactly but only approximately [23]. For this reason, the use of data originating from these system for learning or correcting robot models, it's nowadays a methodology very appreciated in the robotics engineer community. This idea has been performed mainly in two ways: the first and more

classical approach is the so called *model identification*, performed with the use of the linear parametrization of the rigid robots manipulator dynamics (Sec. 3.2); more recently, thanks to the development of algorithms and technologies, model reconstruction has been accounted as a regression problem and solved with machine learning techniques (such as Least Squares and so on). This procedure is called *model learning* (Sec. 3.3.1).

3.2 Linear parametrization of robot dynamics

Considering a fully rigid robot, each link is characterized by 10 dynamic parameters, consisting

- Mass $\rightarrow m$
- Center of mass $\rightarrow \boldsymbol{r}_{\text{cm}} = (r_{\text{cm},x}, r_{\text{cm},y}, r_{\text{cm},z})^T$
- Inertia matrix with respect to the body frame $\rightarrow \boldsymbol{J}_C = \begin{pmatrix} J_{C,xx} & J_{C,xy} & J_{C,xz} \\ & J_{C,yy} & J_{C,yz} \\ \text{symm} & & J_{C,zz} \end{pmatrix}$

The number of dynamic parameters is $10N$, quite high; despite of this, the robot dynamics is dependent only from a subset of them, merged together nonlinearly, called *standard dynamic parameters*. This combination allows to rewrite both the kinetic and potential energy as linear w.r.t. to these new parameters; moreover, since the computation of Lagrange motion equations includes only linear operations (actually derivatives), also these equations will be linear with respect to these parameters

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{Y}_\pi(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})\boldsymbol{\pi} = \boldsymbol{\tau} \tag{3.1}$$

with $\boldsymbol{\pi} = (\boldsymbol{\pi}_1^T \ \boldsymbol{\pi}_2^T \ ... \ \boldsymbol{\pi}_N^T)$ and $\boldsymbol{\pi}_i$ vector composed by

$$\boldsymbol{\pi}_i = (m_i, \ m_i r_{\text{cm}_i,x}, \ m_i r_{\text{cm}_i,y}, \ m_i r_{\text{cm}_i,z}, \ J_{C_i,xx}, \ J_{C_i,xy}, \ J_{C_i,xz}, \ J_{C_i,yy}, \ J_{C_i,yz}, \ J_{C_i,zz}) \tag{3.2}$$

and $\boldsymbol{Y}_\pi \in \mathbb{R}^{N \times 10N}$ called the regressor matrix.

Not all the parameters appears in the dynamic model or just appears in fixed combination; this is the reason for which the regressor has respectively some columns equal to zero or linearly dependent. The reduction of this matrix to just $p < 10N$ fully independent columns allows then to recombine standard parameters to obtain a smaller set called *dynamics coefficients* (or base parameters)

$$\boldsymbol{Y}_\pi \boldsymbol{\pi} \rightarrow \boldsymbol{Y}\boldsymbol{a} \tag{3.3}$$

with $\boldsymbol{Y} \in \mathbb{R}^{N \times p}$. The concept of the model learning consists in the retrieving of these dynamic coefficients through identification procedures. Is critical the fact that the correct estimation of the coefficients doesn't allow in general to recover the dynamic parameters and so the Euler-Lagrange model, since the non-invertibility of the mapping (Fig. 3.1).

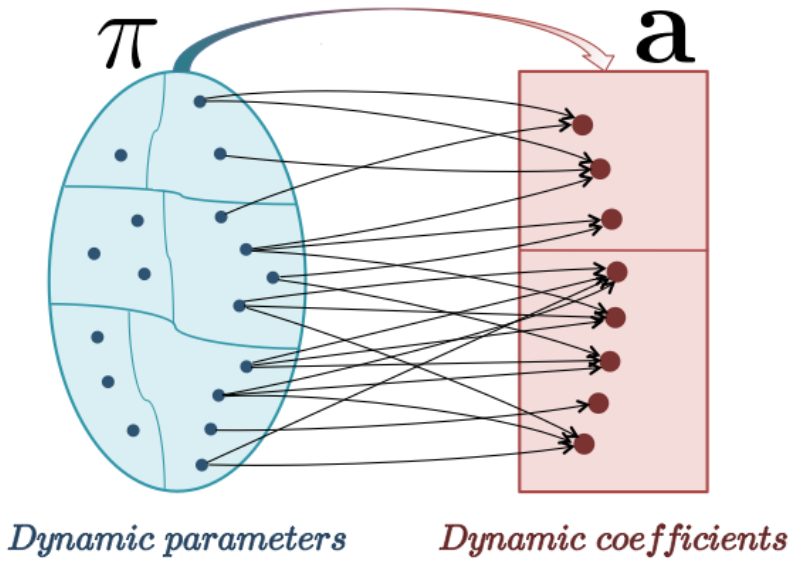

Figure 3.1. The mapping between dynamic parameters and dynamic coefficients is in general nonlinear and not invertible. The use of specific constraints on the optimization, such as in [24], allows to make this mapping again injective (so invertible) and to recover the classic Euler-Lagrange model. Image taken from [24].

3.2.1 Identification

According to what said before, the eq. 3.1 is reformulated as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{Y}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})\boldsymbol{a} = \boldsymbol{\tau} \tag{3.4}$$

In the presence of torque and joint sensors, is then possible to measure joint positions (and consequently compute via numerical differentiation also velocities and accelerations) together with motors torque values. Through a sampling during the robot motion, these quantities are collected and grouped in tuples, each one composed by the regressor (evaluated at the specific time) and the corresponding torque $(\boldsymbol{Y}(\bar{\boldsymbol{q}}, \bar{\dot{\boldsymbol{q}}}, \bar{\ddot{\boldsymbol{q}}}), \bar{\boldsymbol{\tau}})$, linked by the following relation

$$\boldsymbol{Y}(\bar{\boldsymbol{q}}, \bar{\dot{\boldsymbol{q}}}, \bar{\ddot{\boldsymbol{q}}})\boldsymbol{a} = \bar{\boldsymbol{\tau}} \tag{3.5}$$

The stacks of these tuples is used for construct a linear system in the following form

$$\begin{pmatrix} \boldsymbol{Y}(\boldsymbol{q}(t_1), \dot{\boldsymbol{q}}(t_1), \ddot{\boldsymbol{q}}(t_1)) \\ \boldsymbol{Y}(\boldsymbol{q}(t_2), \dot{\boldsymbol{q}}(t_2), \ddot{\boldsymbol{q}}(t_2)) \\ \dots \\ \boldsymbol{Y}(\boldsymbol{q}(t_k), \dot{\boldsymbol{q}}(t_k), \ddot{\boldsymbol{q}}(t_k)) \end{pmatrix} \boldsymbol{a} = \begin{pmatrix} \boldsymbol{\tau}(t_1) \\ \boldsymbol{\tau}(t_2) \\ \dots \\ \boldsymbol{\tau}(t_k) \end{pmatrix} \tag{3.6}$$

in short $\bar{\boldsymbol{Y}}\boldsymbol{a} = \bar{\boldsymbol{\tau}}$, with $\bar{\boldsymbol{Y}} \in \mathbb{R}^{k \times N}$ (k number of samples). The resolution of the system in order to find the vector of dynamic coefficients can be realized in many ways; the most basic approach, effective in case that the rank of the matrix $\bar{\boldsymbol{Y}}$ is at least p, is the pseudoinversion

$$\boldsymbol{a} = \bar{\boldsymbol{Y}}^{\#}\bar{\boldsymbol{\tau}} \tag{3.7}$$

Also other approaches can be employed, such as the setting of an optimization problem of this form

$$\begin{aligned} \min_{\boldsymbol{a}} \quad & \|\bar{\boldsymbol{Y}}\boldsymbol{a} - \bar{\boldsymbol{\tau}}\|^2 \\ \text{subject to} \quad & \boldsymbol{g}(\boldsymbol{a}) \leq \boldsymbol{0} \\ & \boldsymbol{h}(\boldsymbol{a}) = \boldsymbol{0} \end{aligned} \tag{3.8}$$

with $\boldsymbol{a}$ vector of parameters (the decision variable of the problem), a quadratic loss representing the square error between the measured and predicted torque and $\boldsymbol{g}(\boldsymbol{a})$ and $\boldsymbol{f}(\boldsymbol{a})$ inequalities/equalities constraints, inserted to introduce external information around the system, such as the expected range of some parameters or physically based considerations (e.g only positive values for the masses of the links) [24]. The steps aforementioned are usually employed offline and can be summarized as follows:

- Trajectories definition
- Trajectory tracking (through decentralized controller such as PD on single joint) and contemporary data sampling from sensors (encoder/torque sensors)
- Offline identification (resolution of the linear system)

In order to assure high likelihood of the parameters obtained through the identification procedures, the trajectories executed by the robot (from which the data are extracted) should present some features. These kind of trajectories, that preserve a certain level of information during the motion, are called *Persisting Exciting Trajectories*. Formally speaking, giving a signal represented by the regressor matrix $\boldsymbol{Y}$, the reference trajectory $\boldsymbol{q}_d$ and $\boldsymbol{Y}_d$ the regressor evaluated on the sampled trajectory points, the persistency condition is satisfied if, for all $t_1 \geq 0$, there exist $\delta, \alpha_1, \alpha_2$ positive constants such that [25]

$$\boldsymbol{I}\alpha_1 \geq \int_{t_1}^{t_1+\delta} \boldsymbol{Y}_d^T \boldsymbol{Y}_d dt \leq \boldsymbol{I}\alpha_2 \tag{3.9}$$

This condition is necessary to reject the noise measurements and improve the reconstruction; moreover, can be shown that this inequality is equivalent to the minimization of the condition number of the regressor matrix. Generation of trajectories that satisfy the persistent excitation can be realized choosing sinusoidal motions with a wide range of frequencies and amplitudes or, in a more accurate way, setting a proper optimization problem [26].

Strictly speaking the identification step may be realized also inline, through techniques such as recursive least square or gradient based; still, since not explicitly finalized to control the system, is not really advantageous.

Ultimately should be highlighted that the identification step allows to recover dynamic coefficients, that, as was said in previous sections, do not coincide with dynamic parameters. This implies that they cannot directly used for recovering the Euler-Lagrange and so employ aforementioned model based controller (such as Feedback Linearization). A further step to identify the actual physical parameters must be executed.

Assuring the the physical consistence of the coefficients is almost mandatory and convenient in these procedures, since the actual minimization of the prediction error without assuring specific constraints can produce solutions quite unrealistic (e.g negative masses) and not useful to recover the actual model of the robot [24].

3.2.2 Identification from motor currents with unknown signs

As said before, robot dynamic identification techniques rely on the quality and completeness of the signals available as inputs, typically joint positions and motor currents or joint torques. These signals are often noisy and filtering operations are required before using them for identification. An interesting problem is to realize a correct model estimation when the robot returns only the absolute values of the motor currents (or of the torques). Let's consider the eq. 3.5, modified to link directly the dynamic model to the current absorbed by the motors (and not anymore to the torque). In general the relationship between the torque generated by the motor and the current is not simple, but in first approximation is considered as linear with a gain specific for each motor, summarized as a diagonal positive definite matrix $\boldsymbol{K}$

$$\boldsymbol{\tau} = \boldsymbol{K}_I \boldsymbol{i} \tag{3.10}$$

with $\boldsymbol{i}$ current absorbed by the motors and available for measurements. This complication doesn't prevent the of aforementioned methods; what is done is in fact a first step of estimation of these gains (through static identification procedures) and then the pseudoinversion of a modified regressor matrix [2] (the original one pre-multiplied to the inverse of the gain matrix)

$$\tilde{\boldsymbol{Y}} = \boldsymbol{K}_I^{-1} \boldsymbol{Y} \boldsymbol{a} = \boldsymbol{i} \tag{3.11}$$

Nevertheless, if the measurements of the motor currents present only their absolute value and not the sign, this cannot be straightly applied. In general this is not a common situation (nowadays modern complaint robot provides complete torque measurements) but old platform such as KUKA KR5 Sixx or Nao Humanoid robot still presents this issue. A way to overcome this problem was proposed in [27]. The main idea of this work is that, assuming the continuity of the torque signals, is possible to split current measurement in segments, each one interrupted when the absolute value reaches a predefined threshold around zero. The choice of the sign for these segments is then formalized as an informed search on a tree, in which every leaf corresponds to a possible sign configuration of these segments, having 3.9 as cost function. This tree is recursively built: first of all, a starting initial set of measurements with a given sign is arbitrarily chosen; then new measurements are inserted and used for solving many optimization problems, with all possible combinations of the currents signs. After that, the combination that minimizes the loss function is considered as correct and introduced as a node in this tree. This procedure is then iterate many times and terminates when is defined the sign for all the acquired data. Finally the last step is the resolution of the classic identification problem, obtained after that to each segments is assigned the corresponding sign. This heuristic proved to be very efficient: the built of the dynamic model in fact was so precise that allowed to implement human-robot interaction techniques, such

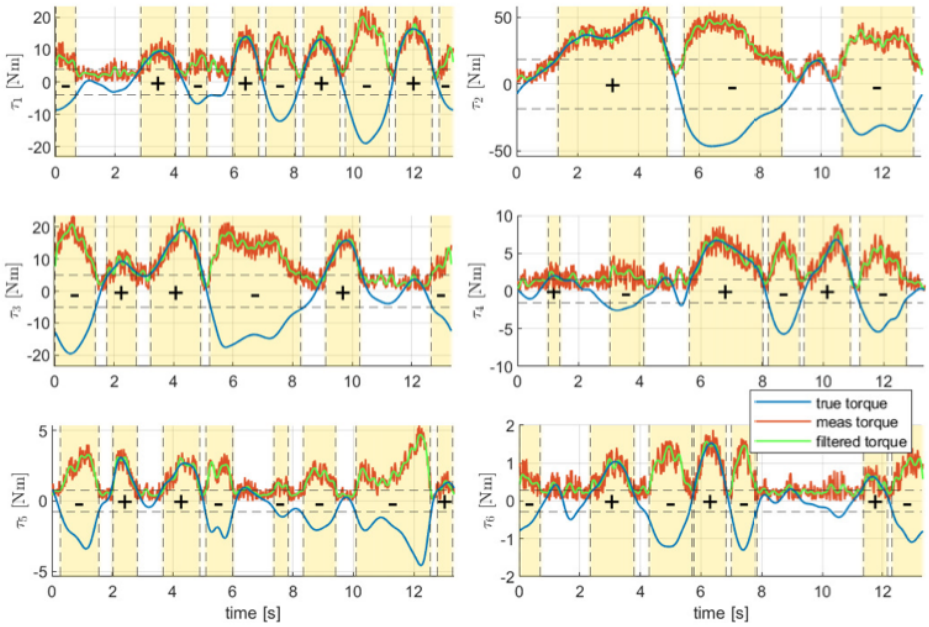

Figure 3.2. Simulated torques for the KUKA KR5 robot during an exciting trajectory. The nominal torques (blue lines) are hidden to the user, while only their absolute values (with additional noise) are available as measures (red lines). These are filtered (green lines) before being used in the optimization. The yellow strips highlight the detected segments, and the +/- symbols indicate the signs estimation obtained as a result of the algorithm. Image taken from [27].

as residual signal evaluation, usually unsuitable for old industrial robots (KUKA KR5 Sixx).

3.2.3 Adaptive control

Another way to face the trajectory tracking problem is to design a nominal controller which includes as further effect the online adaptation of the dynamics coefficients. The paradigm in this case shifts from just model identification to identification for control. The goal becomes in fact the zeroing of the tracking error and not anymore the precise reconstruction of the robot model. A classic of this is the so called *Adaptive Control*, in which the control law includes the dynamics coefficients as part of the state and provides a virtual update dynamics. In detail, given a reference trajectory $\boldsymbol{q}_d$, is possible to define the vector of position error as $\tilde{\boldsymbol{q}} = \boldsymbol{q}_d - \boldsymbol{q}$ and a variable $\boldsymbol{q}_r$, such that $\dot{\boldsymbol{q}}_r = \dot{\boldsymbol{q}}_d + \boldsymbol{\Lambda}\tilde{\boldsymbol{q}}$ (with $\boldsymbol{\Lambda} > 0$). Roughly speaking, the velocity of this new variable increases if the position error increments. Considering what said before and defining the nominal dynamics with the same notation but with the hat on it, the adaptive controller is characterized by the following control law:

$$\begin{aligned} \boldsymbol{\tau}_{adaptive} &= \hat{\boldsymbol{M}}(\boldsymbol{q})\ddot{\boldsymbol{q}}_r + \hat{\boldsymbol{C}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}_r + \hat{\boldsymbol{g}}(\boldsymbol{q}) - \boldsymbol{K}_D\boldsymbol{s} \\ \dot{\hat{\boldsymbol{a}}} &= -\boldsymbol{\Gamma}\boldsymbol{Y}\boldsymbol{s} \end{aligned} \tag{3.12}$$

where $\boldsymbol{\Gamma}$ and $\boldsymbol{K}_D$ positive definite matrices and $\boldsymbol{s} = \dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_r = \dot{\tilde{\boldsymbol{q}}} + \boldsymbol{\Lambda}\tilde{\boldsymbol{q}}$ error surface. Splitting the term $\boldsymbol{K}_D\boldsymbol{s} \rightarrow \boldsymbol{K}_D\dot{\tilde{\boldsymbol{q}}} + \boldsymbol{K}_D\boldsymbol{\Lambda}\tilde{\boldsymbol{q}}$ is evident that the gains $\boldsymbol{K}_D$ and $\boldsymbol{K}_D\boldsymbol{\Lambda}$

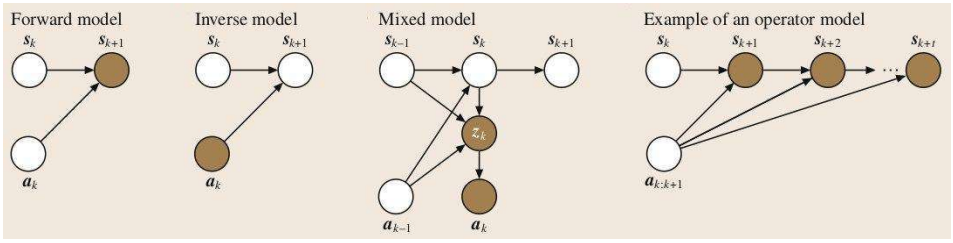

Figure 3.3. In the picture are represented different type of model learning, assigning white color to the circle representing the available quantities and using instead the dark brown to characterize the unknown variables. From left to right: Forward model, in which the mapping to reconstruct is $(s_k, a_k) \rightarrow s_{k+1}$); Inverse Model, in which instead the relationship is $(s_k, s_{k+1}) \rightarrow a_k$). Third from left is presented the scheme of Mixed Model learning, in which forward and inverse model learning are jointly used in order to add consistency to the reconstruction. Eventually the operator model learning is showed; in this case, the goal is to understand the evolution of the steps n-steps ahead via the reconstruction of the time evolution operator of the system. Image: © Bruno Siciliano, Oussama Khatib, *Springer Handbook of Robotics*, 2008, Chapter 15

are assigned respectively to compensate position and velocity errors. The above control and adaptation law assures global convergence of the tracking error (as long as the reference trajectory is bounded) [25]. However stability of the error dynamics doesn't imply convergence of the dynamics coefficients to the real values; in fact, as for the identification task, in general persisting exciting trajectories are still a mandatory requirement. The strength of the approach has been validated on different platforms; nevertheless, the area of application is very narrow, since requiring many assumptions. To extend the area of applications and relax some assumptions (some of which quite unrealistic for the application in a real context), data driven techniques have been exploited.

3.3 Machine learning approaches

3.3.1 Model learning in robotics

An accurate knowledge of the robot's model is quite significant aspect, since allows to design an effective controller, to predict the behaviour of the system and delineate a correct interaction with its environment. This can be realized in many ways, according to the mathematical formulation of the problem. As said in the introductory chapter, according to the quantities that are available, different methods can be employed. Let's consider a discrete system described at each time step t_k with a state variable s_k. According to its specificity, the state can be just the generalized coordinates (such as in Lagrange formulation), some outputs (e.g. a subset of the complete state) or some variables defined in a more abstract way. In this characterization of the system evolution, at each step the robot (considered as an autonomous agent) execute a specific action a_k that drives the system to the next state s_{k+1}. Schematically, this can be represented as

$$(s_k, a_k) \rightarrow s_{k+1} \tag{3.13}$$

This general description includes system represented by ordinary differential equations, such as robots (using well known discretization techniques) but also autonomous agents dynamics, in a more computer science oriented setting. The direction of the arrow illustrate the casual relationships between the two states; in classical mechanics this mappings is unique, since the tuple current state-action encapsulates the necessary information to determinate the time evolution. In general this is not an absolute property [28]. If in the description are included also stochastic effects, such as random noises, a probabilistic model is a more suitable approach to capture the system's evolution. In particular, a probability density function (PDF) can be used to expresses the likelihood of a given next state conditioned to the previous state-action tuple

$$T(s_{k+1}, s_k, a_k) = P(s_{k+1} \| s_k, a_k) \tag{3.14}$$

In this description states are random variables and, according to the assumption realized on the noise acting on it, different PDF are associated with the random process. In this formalism *forward model learning* is expressed as the approximation of this mapping (both in the deterministic or random case) through another function, usually parameterized. Application of these approaches to control robotic sytems are copious. In [29] the authors proposed a Model Predictive Control approach able to merge robust MPC approaches (Tube MPC) while improving the forward model of the system through learning algorithms. This is realized with a complete decoupling between them: constraints are enforced considering the worst case effects of unmodeled dynamics while the cost function optimization is instead realized with the learned linear model. Another classic approach is the one developed in [11], in which a probabilistic model of the forward dynamics of a mobile robot is learnt. The peculiarity of this approach is the use of bayesian techniques, such as Gaussian process and Bayesian Linear Regressors, that allows the quantification of the uncertainty in the reconstructed model. This is similar to what was done in [30] for the case of a fully actuated robot manipulator. In general is evident how the use of model based controllers, that exploits the transition model to generate a feedback (such as Model Predictive Controller), is very advisable and one of the major application of this methodology.

On the other hand, in some contexts such as task of trajectory tracking, the next state is just given (e.g. a point on the reference trajectory) and the goal is to find the action able to steer the current state to the desired one. The reconstruction of the mapping that, given current and next state, provides the required action is called *inverse model learning*. Formally speaking the mapping is expressed as

$$(s_k, s_{k+1}) \to a_k \tag{3.15}$$

This relationship is not physically grounded, since represents an anticausal system; this in general causes the ill-posedness of the problem. Although, inverse dynamics learning of a fully actuated problem is a valid application of this method, while for redundant robot this issue must be properly managed. In agreement what said in previous chapters, since the target mapping have as output an action, it may be classified as control learning. This separation is not so rigid; as example, for the case of feedback linearization learning, in which the action corresponds to a torque that

compensates the model, there is an overlapping between the control input and the model. A glaring example of this is the FL learning with the assumption of a precise nominal inertia matrix, so having $\boldsymbol{M} = \hat{\boldsymbol{M}}$ (in general an acceptable assumption). In this case the extra term in eq. 2.16, that need to be learned, consists in the missing Coriolis and Gravity or friction terms; then the inverse model learning is also a forward model learning. Many works have been realized on this topic. In [31] is proposed the improvement of a computed torque controller through the learning of the feedforward nonlinear term (forward model learning) and then an inverse model learning is realized to approximate the mapping providing the torque input to feedback linearize the system. Another similar approach is [32], that similarly uses Bayesian approaches (in particular Gaussian Processes) to estimate the missing part of the model and evaluate the reconstruction error; indeed, exploiting known probabilistic bounds on reconstruction, the author proves the ultimately boundness of the error dynamics. Again the idea of the learning of the missing FL terms while assuring convergence of the closed loop dynamics is realized in [33]. Thus, differently from Bayesian approaches, parametric structure of the approximator (in this case a Neural Network) is investigated and used for defining a suitable Lyapunov function and then guarantee stability. An important hypothesis of the inverse model learning scheme is the fact that this relationship is unique. This is not always true; a representative example is the inverse kinematic of a redundant robot that, as said before in Ch. 2, is a mapping lacking of the injectivity property. In fact, for a given task value, there is an entire subspace of the Configuration Space representing it. In this context Mixed models learning methods are employed. This technique is a mix between forward and inverse model learning and is also called *distal learning*. It consists in firstly approximating the forward model and then the inverse model of the same process. In such a way the inverse model will minimize the error with respect both to the output of the direct model and the input of the inverse one. Ideally, in the case of a perfect reconstruction, the composition of the two mapping should result in the identity operator. A famous application of this approach in robotics is [34], in which the distal learning algorithm is introduced for the first time in literature and tested on simulated robotic system in order to recover both kinematics and dynamics modeling.

Eventually, a more modern model learning techniques is introduced, called *operator learning*. It consists in the reconstruction of the time evolution operator, the operator that applied to the system state (or to an output) at time t_k provides its value at the time t_{k+n}, so its general time evolution and not only one step ahead. Despite the fact that the formulation seems quite similar to forward model learning, it is conceptually very different. In fact the focus in this approach is not the recover of a punctual approximation of the dynamics; instead the purpose is to recover the general operator representation (in a suitable space, according to the system and from the particular mathematical scheme). In general the representation of this operator is not linear and so hard to retrieve, but a new promising approach (in particular for system which modeling is really imprecise such as soft robots, multi body constrained dynamics and so on) is the use of Koopman spectral theory, in which nonlinear dynamics are represented in terms of an infinite dimensional linear operator acting on the space of all possible output (observable) of the system [35]. Since the operator in this new coordinates is represented as linear (at the cost of

infinite dimensionality), classic techniques used for infinite linear operator reduction (such as dynamic mode decomposition [36]) can be used to retrieve a good data driven approximation. Eventually, this linear operator can be used to construct optimal controller, such as Model Predictive Control or Linear Quadratic Regulator, also for very complex nonlinear systems [37].

All these mechanisms are based on the idea that a relationship, in which input and output spaces are defined, must be reconstructed (at least approximately) using its instances, collected in a set called dataset. This is called in machine learning a regression problem, which is going to be formulated clearly in the next section.

3.3.2 The regression problem

Let's consider a generic mapping between two space (X, Y). According to the properties that this general mapping satisfies can be named in many ways such as curve, function and so on. In this case, the only guess around this mapping is the fact that exists an independent variable X, called conventionally input, and a dependent variable, called output Y, which specifies the direction of the mapping (the casual relationship) from $X \to Y$. Conforming to what just said, a representation of this mapping is not yet defined. In general is not necessary since the regression problem consists in the approximation of this mapping through only the use of its instances. Practically speaking, is assumed to have enough samples of this relationship (collected in a structure such as a table) such that a model of the process is reconstructed, via an inductively reasoning, able to approximate the map well enough. According to the clarification of what approximation means, different methods for realizing this goal are employed. Since the extent of the topic, many mathematical formulation have been employed, from statistically framework to optimization settings; they are all valid and sometimes useful to observe the same thing in a different way. The first approach considered for solving a regression problem is least square methods (LS). The main concept is firstly the representation of the mapping through a parametric model and then the finding of the optimal parameters (weights) $\boldsymbol{\theta}_{LS}$ that better minimize the following cost function

$$\begin{aligned} \boldsymbol{\theta}_{LS} &= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \|\boldsymbol{y}_i - \boldsymbol{f}_i(\boldsymbol{x}_i, \boldsymbol{\theta})\|^2 = \\ &\arg\min_{\boldsymbol{\theta}} \left(\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})\right)^T \left(\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})\right) \end{aligned} \tag{3.16}$$

where $\boldsymbol{y}$ is the instance of the mapping, $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})$ is the parametric representation and $i = 1...N$ is the number of tuples (instances) collected. Is called least square problem since the cost function is the square norm of the residual vector $\boldsymbol{r} = \boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})$, which represents the difference between the predicted mapping computed in $\boldsymbol{x}$ and the actual value $\boldsymbol{y}$. In case of a specific model, such as nonlinear with respect to the input but linear in the weights

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\theta} \tag{3.17}$$

the solution of the problem has closed form. In fact, stacking the instances of the mapping in a matrix $\boldsymbol{Y}$, the linear system can be solved through the operator

formerly presented, the Pseudoinverse matrix (2.26)

$$\boldsymbol{Y} = \begin{pmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \dots \\ \boldsymbol{y}_n \end{pmatrix} = \begin{pmatrix} \boldsymbol{\phi}_1 \\ \boldsymbol{\phi}_2 \\ \dots \\ \boldsymbol{\phi}_n \end{pmatrix}^T \boldsymbol{\theta} = \boldsymbol{\Phi}^T \boldsymbol{\theta} \rightarrow \tag{3.18}$$

$$\boldsymbol{\theta}_{LS} = (\boldsymbol{\Phi}^T)^{\#} \boldsymbol{Y}$$

Can be clearly recognized the formulation previously introduced in 3.7, that is in fact a particular example of a larger class of least square regression problem. According to how the approximation model is mathematical represented, different methods must be used. An elegant method for solving the problem in case of nonlinear dependency of the model with respect to the parameters, is the so called *Gauss-Newton algorithm* [38]. Also in this condition a LS problem is formulated but, differently from before, the model representing the mapping $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})$ is not anymore linear with respect to the parameters but just a general nonlinear function (assumed to be differentiable). Reminding the characterization of the residual vector as $\boldsymbol{r} = \boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})$, is possible in fact to define the Jacobian matrix $\boldsymbol{J}_r = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\theta}}$ and the initial guess for the parameters (as solution of the least square problem) $\boldsymbol{\theta}_0$. Then, using the Taylor's approximation of the residual around this starting hypothesis

$$\boldsymbol{r}(\boldsymbol{\theta}) = \boldsymbol{r}(\boldsymbol{\theta}_0) + \boldsymbol{J}_r^T(\boldsymbol{\theta}_0)\boldsymbol{\delta} + \mathcal{O}(\|\boldsymbol{\delta}\|^2) \tag{3.19}$$

with $\boldsymbol{\delta} = (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$ the originally nonlinear least square problem is transformed into a linear one

$$\boldsymbol{\delta}_{GN} = \arg\min_{\boldsymbol{\delta}} \ \|\boldsymbol{r}(\boldsymbol{\theta}_0) + \boldsymbol{J}_r^T(\boldsymbol{\theta}_0)\boldsymbol{\delta}\|^2 \tag{3.20}$$

$$\tag{3.21}$$

which is known to have a closed form solution (3.19), valid in the neighborhood of the linearization point. This procedure is iterated until the convergence (in which the Jacobian is almost flat). Least square approaches are intuitive, understandable and very effective; still they not include any stochastic effects. A formulation more realistic of the same approach that includes also random disturbances is obtained through the so called *Maximum likelihood estimation*. Let's consider, in the same notation as before, the target deterministic function to reconstruct. A Gaussian noise $\boldsymbol{\zeta}$ is added such that every instance of this mapping will be

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}) + \boldsymbol{\zeta} \tag{3.22}$$

The assumption of Gaussian (also called Normal) probability distribution for representing a continuous process noise is quite common. Given a noise signal $\boldsymbol{\zeta} \in \mathbb{R}^D$, its usually representation is in fact a normal distribution having zero mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ (symmetric and positive definite)

$$\mathcal{N}(\boldsymbol{\zeta}\|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{\zeta} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{\zeta} - \boldsymbol{\mu})\right) \tag{3.23}$$

Since the target mapping was assumed deterministic, the sum of a deterministic process and a Gaussian noise will result in a random variable normally distributed as follows

$$P(\boldsymbol{y}\|\boldsymbol{x},\boldsymbol{\theta},\boldsymbol{\Sigma}) = \mathcal{N}(\boldsymbol{y}\|\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}),\boldsymbol{\Sigma}) \tag{3.24}$$

The data representing the instances of this mapping, previously collected as tuples in the matrix $\boldsymbol{Y}$, can be considered as randomly extracted via the probability distribution represented in eq. 3.24, each one independently from the other. This is equivalent to say that

$$P(\boldsymbol{Y}\|\boldsymbol{X},\boldsymbol{\theta},\boldsymbol{\Sigma}) = P(\boldsymbol{y}_1,\boldsymbol{y}_2,...,\boldsymbol{y}_N\|\boldsymbol{x}_1,\boldsymbol{x}_2,...,\boldsymbol{x}_N,\boldsymbol{\theta},\boldsymbol{\Sigma}) = \prod_{i=1}^{N} P(\boldsymbol{y}_i\|\boldsymbol{x}_i,\boldsymbol{\theta},\boldsymbol{\Sigma}) \tag{3.25}$$

with $P(\boldsymbol{Y}\|\boldsymbol{X},\boldsymbol{\theta},\boldsymbol{\Sigma})$ called likelihood function, function representing the probability of the collected instances of the process given the input, the noise's variance and, most important, the assigned set of parameters $\boldsymbol{\theta}$. Ideally speaking the optimal set of weights that should be used to represent the process are the ones who maximizes the likelihood function. This is the core of the maximum likelihood estimation approach

$$\boldsymbol{\theta}_{MLE} = \arg\max_{\boldsymbol{\theta}} \; P(\boldsymbol{Y}\|\boldsymbol{X},\boldsymbol{\theta},\boldsymbol{\Sigma}) \tag{3.26}$$

An important feature of this formulation is the fact that logarithm function can be applied on the loss function still preserving the result of the optimization

$$\arg\max_{\boldsymbol{\theta}} \; P(\boldsymbol{Y}\|\boldsymbol{X},\boldsymbol{\theta},\boldsymbol{\Sigma}) = \arg\max_{\boldsymbol{\theta}} \; \ln(P(\boldsymbol{Y}\|\boldsymbol{X},\boldsymbol{\theta},\boldsymbol{\Sigma})) \tag{3.27}$$

Merging eq. 3.27 and eq. 3.25 the results is

$$\begin{aligned} &\sum_{i=1}^{N} \ln(P(\boldsymbol{y}_i\|\boldsymbol{x}_i,\boldsymbol{\theta},\boldsymbol{\Sigma})) = \\ &-\frac{D}{2}\ln(2\pi) - \frac{1}{2}\ln(|\boldsymbol{\Sigma}|) + \frac{1}{2}\sum_{i=1}^{N}\Big((\boldsymbol{y}_i - \boldsymbol{f}_i(\boldsymbol{x}_i,\boldsymbol{\theta}))^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{y}_i - \boldsymbol{f}_i(\boldsymbol{x}_i,\boldsymbol{\theta}))\Big) \end{aligned} \tag{3.28}$$

and so the optimization problem becomes a Weighted Least Square problem

$$\boldsymbol{\theta}_{MLE} = \arg\min_{\boldsymbol{\theta}} \; (\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}))^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta})) \tag{3.29}$$

in which arg min becomes arg max since the change of sign before the sum. The problem is very similar to the one in 3.16, but the norm is weighted through the inverse of the covariance matrix. Assuming a variance diagonal and unitary $\boldsymbol{\Sigma} = \boldsymbol{I}$ the Maximum Likelihood Estimation coincides with the Least Square estimation $\boldsymbol{\theta}_{LS} = \boldsymbol{\theta}_{MLE}$. Again, in case of a linear relationship of the process with respect to the parameters (as in 3.17) and a generic covariance matrix the solution will be

$$\boldsymbol{\theta}_{MLE} = (\boldsymbol{\Phi})^{\#}_{\boldsymbol{\Sigma}^{-1}}\boldsymbol{Y} \tag{3.30}$$

This weighing has a clear heuristic interpretation: since the inverse of the covariance matrix is the information matrix, representing the amount of precision around

some measurements, components of a low magnitude (so low information) are less considered in the estimation of the optimal parameters. These two basic approaches are some of many that can be used to solve a regression problem. In next section another approach, partially related to the Maximum Likelihood Estimation will be presented and deeply investigated. In fact it will represent the primary method employed in the research results delineated in this manuscript.

3.3.3 Gaussian process regression

In this section is introduced one of the main tools used for the development of the novelties presented in this thesis. As for the approaches, the main idea is to deal with the problem of the robot dynamics reconstruction through the formulation of a regression problem. As well as Maximum Likelihood Estimation, this mapping is defined as a stochastic process, introducing a random noise acting on it. Still the following approach, introduced with the same premise as MLE, differs largely from it. Let's consider the eq. 3.22 with the assumption of scalar output so having $y \in R$ while having the input vector of dimension n_x so $\boldsymbol{x} \in \mathbb{R}^{n_x}$. Again the mapping is approximated through a parametric probabilistic distribution using the weights vector $\boldsymbol{\theta}$. The approach now diverges from the previous one; in fact, instead of trying to maximize $P(\boldsymbol{Y} \| \boldsymbol{X}, \boldsymbol{\theta}, \boldsymbol{\Sigma})$, so the likelihood of the measured instances of the process conditioned to the given input and parameter's set, the new target is the maximization of the posterior probability of the parameters, so $P(\boldsymbol{\theta} \| \boldsymbol{Y}, \boldsymbol{X}, \boldsymbol{\Sigma})$. This technique is called *Maximum a Posteriori Estimation* (MAP). In order to explicit this posterior probability can be used Bayes's theorem

$$P(\boldsymbol{\theta} \| \boldsymbol{Y}, \boldsymbol{X}, \boldsymbol{\Sigma}) = \frac{P(\boldsymbol{Y} \| \boldsymbol{X}, \boldsymbol{\theta}, \boldsymbol{\Sigma}) P(\boldsymbol{\theta})}{P(\boldsymbol{Y} \| \boldsymbol{X}, \boldsymbol{\Sigma})} \tag{3.31}$$

Since the denominator is independent from the parameters is irrelevant and so the optimization problem becomes the follow

$$\boldsymbol{\theta}_{MAP} = \arg\max_{\boldsymbol{\theta}} \; P(\boldsymbol{Y} \| \boldsymbol{X}, \boldsymbol{\theta}, \boldsymbol{\Sigma}) P(\boldsymbol{\theta}) \tag{3.32}$$

In order to find the optimal set of parameters, an adding information with respect to 3.26 is required: the prior probability distribution of the parameters $P(\boldsymbol{\theta})$. Again a classical probabilistic prior assigned to these parameters is a normal distribution with zero mean $\mu_{\boldsymbol{\theta}}$ and covariance matrix $\Sigma_{\boldsymbol{\theta}}$

$$P(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} \| \mathbf{0}, \Sigma_{\boldsymbol{\theta}}) \tag{3.33}$$

A way to explicitly use this method is, as before, assuming the linearity with respect to the parameters of the model, so (as shown in eq. 3.17) $y = \phi(\boldsymbol{x})^T \boldsymbol{\theta}$ (and $\boldsymbol{Y} = \boldsymbol{\Phi}(\boldsymbol{X})^T \boldsymbol{\theta}$, considering all the instances of the relationship). As before, given the Gaussianity of $P(\boldsymbol{Y} \| \boldsymbol{X}, \boldsymbol{\theta}, \boldsymbol{\Sigma})$ and $P(\boldsymbol{\theta})$ and the fact the scalar output reduces also the covariance matrix $\boldsymbol{\Sigma}$ to a scalar vector $\boldsymbol{\Sigma} \to \sigma^2$

$$\begin{aligned} &P(\boldsymbol{\theta} \| \boldsymbol{Y}, \boldsymbol{X}, \boldsymbol{\Sigma}) = P(\boldsymbol{Y} \| \boldsymbol{X}, \boldsymbol{\theta}, \boldsymbol{\Sigma}) P(\boldsymbol{\theta}) \approx \\ &\exp\left(-\frac{1}{2\sigma^2}(\boldsymbol{Y} - \boldsymbol{\Phi}(\boldsymbol{X})^T \boldsymbol{\theta})^T (\boldsymbol{Y} - \boldsymbol{\Phi}(\boldsymbol{X})^T \boldsymbol{\theta})\right) \exp\left(-\frac{1}{2}\boldsymbol{\theta}^T \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} \boldsymbol{\theta}\right) \to \\ &\exp\left(-\frac{1}{2}(\boldsymbol{\theta} - \bar{\boldsymbol{\theta}})^T (\frac{1}{2\sigma^2} \boldsymbol{\Phi}(\boldsymbol{X}) \boldsymbol{\Phi}(\boldsymbol{X})^T + \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1})(\boldsymbol{\theta} - \bar{\boldsymbol{\theta}})\right) \end{aligned} \tag{3.34}$$

with $\bar{\boldsymbol{\theta}}(\boldsymbol{Y}, \boldsymbol{X}) = \sigma^{-2}\left(\sigma^{-2}\boldsymbol{\Phi}(\boldsymbol{X})\boldsymbol{\Phi}(\boldsymbol{X})^T + \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\right)^{-1}\boldsymbol{\Phi}(\boldsymbol{X})\boldsymbol{Y}$. Can be recognized the form of a normal distribution with mean vector $\bar{\boldsymbol{\theta}}$ and covariance matrix $\boldsymbol{A} = (\frac{1}{2\sigma^2}\boldsymbol{\Phi}(\boldsymbol{X})\boldsymbol{\Phi}(\boldsymbol{X})^T + \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1})^{-1}$.

$$P(\boldsymbol{\theta}\|\boldsymbol{Y}, \boldsymbol{X}, \boldsymbol{\Sigma}) = \mathcal{N}(\boldsymbol{\theta}\|\bar{\boldsymbol{\theta}}, \boldsymbol{A}) \tag{3.35}$$

One more time Bayesian architecture allows to predict the probability of the outcome for a specific test case input $\boldsymbol{x}_*$. Given the corresponding deterministic output $f_* = f(\boldsymbol{x}_*)$ is possible to determine the predictive distribution through the integration of the random parameters $\boldsymbol{\theta}$, a procedure called in statistics marginalization

$$\begin{aligned} &P(f_*\|\boldsymbol{x}_*, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\Sigma}) = \int P(f_*\|\boldsymbol{x}_*, \boldsymbol{\theta})P(\boldsymbol{\theta}\|\boldsymbol{Y}, \boldsymbol{X}, \boldsymbol{\Sigma})d\boldsymbol{\theta} = \\ &\mathcal{N}(f_*\|\frac{1}{\sigma^2}\boldsymbol{\phi}(\boldsymbol{x}_*)^T\boldsymbol{A}\boldsymbol{\Phi}(\boldsymbol{X})\boldsymbol{Y}, \boldsymbol{\phi}(\boldsymbol{x}_*)^T\boldsymbol{A}\boldsymbol{\phi}(\boldsymbol{x}_*)) \end{aligned} \tag{3.36}$$

this can be rewritten with a terminology slightly different but useful for exploring later a new point of view. Let's short the formulation renaming the following quantities: $\boldsymbol{\Phi}(\boldsymbol{X}) = \boldsymbol{\Phi}$, $\boldsymbol{\phi}(\boldsymbol{x}_*) = \boldsymbol{\phi}_*$, $\boldsymbol{K} = \boldsymbol{\Phi}^T\boldsymbol{\Sigma}_{\boldsymbol{\theta}}\boldsymbol{\Phi}$ Then, using some known properties of matrices inversion and some algebraic computation [39] is possible to rearrange the predictive distribution of f_* as

$$\begin{aligned} &P(f_*\|\boldsymbol{x}_*, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\Sigma}) = \\ &\mathcal{N}(f_*\|\boldsymbol{\phi}_*^T\boldsymbol{\Sigma}_{\boldsymbol{\theta}}\boldsymbol{\Phi}(\boldsymbol{K} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{Y}, \boldsymbol{\phi}_*^T\boldsymbol{\Sigma}_{\boldsymbol{\theta}}\boldsymbol{\Phi} - \boldsymbol{\phi}_*^T\boldsymbol{\Sigma}_{\boldsymbol{\theta}}\boldsymbol{\Phi}(\boldsymbol{K} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{\Phi}^T\Sigma_{\boldsymbol{\theta}}\boldsymbol{\phi}_*) \end{aligned} \tag{3.37}$$

Some important peculiarities should be noticed. First of all, the $\boldsymbol{\Phi}(\boldsymbol{X})$ can be considered a transformation from the input space to a more general one, called *Features Space* (commonly with a larger dimension). Moreover, looking at the form of both mean vector and covariance matrix, this features spaces enters in the form of $\boldsymbol{\Phi}^T\boldsymbol{\Sigma}_{\boldsymbol{\theta}}\boldsymbol{\Phi}$, $\boldsymbol{\phi}_*^T\boldsymbol{\Sigma}_{\boldsymbol{\theta}}\boldsymbol{\Phi}$ or $\boldsymbol{\phi}_*^T\boldsymbol{\Sigma}_{\boldsymbol{\theta}}\boldsymbol{\phi}_*$. On top of this is defined a new function, called *Kernel Function*, as

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\Sigma}_{\boldsymbol{\theta}}\boldsymbol{\phi}(\boldsymbol{x}') \tag{3.38}$$

This function represents a scalar product in a suitable space; in fact, reminding that the covariance matrix is for definition positive definite, is possible to calculate its Cholesky Decomposition (considering only real matrices) $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{1/2}$ such that $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{1/2}(\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{1/2})^T$ (equivalent also to the transpose, since the symmetry of the covariance matrix). So substituting this decomposition in eq. 3.38 what happens is that

$$\begin{aligned} &k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T(\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{1/2})^T\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{1/2}\boldsymbol{\phi}(\boldsymbol{x}') \rightarrow \\ &k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\psi}(\boldsymbol{x})^T\boldsymbol{\psi}(\boldsymbol{x}') \end{aligned} \tag{3.39}$$

with $\boldsymbol{\psi}(\boldsymbol{x}) = \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{1/2}\boldsymbol{\phi}(\boldsymbol{x})$. This shows clearly that in this new features space $\boldsymbol{\psi}$ the kernel function represents a scalar product. In this sense, the choice of a particular kernel with respect to another modifies the representative power of the Gaussian

process. A way to represent shortly and intuitively the moments (mean and variance) of the Gaussian distribution in eq. 3.37 is the follow

$$\begin{aligned} \mu_{f^*} &= \boldsymbol{k}(\boldsymbol{x}_*, \boldsymbol{X}) \left(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I} \right)^{-1} \boldsymbol{Y} \\ \sigma^2_{f^*} &= -k(\boldsymbol{x}_*, \boldsymbol{x}_*) \boldsymbol{k}(\boldsymbol{x}_*, \boldsymbol{X}) \left(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I} \right)^{-1} \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_*) \end{aligned} \tag{3.40}$$

with $\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X})$ the kernel matrix defined as

$$\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & k(\boldsymbol{x}_1, \boldsymbol{x}_3) & ... & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & k(\boldsymbol{x}_2, \boldsymbol{x}_3) & ... & k(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ ... & ... & ... & ... & \\ k(\boldsymbol{x}_k, \boldsymbol{x}_1) & k(\boldsymbol{x}_k, \boldsymbol{x}_2) & k(\boldsymbol{x}_k, \boldsymbol{x}_3) & ... & k(\boldsymbol{x}_k, \boldsymbol{x}_N) \end{pmatrix} \tag{3.41}$$

and the vector $\boldsymbol{k}(\boldsymbol{x}_*, \boldsymbol{X})$ as

$$\boldsymbol{k}(\boldsymbol{x}_*, \boldsymbol{X}) = \begin{pmatrix} k(\boldsymbol{x}_*, \boldsymbol{x}_1) & k(\boldsymbol{x}_*, \boldsymbol{x}_2) & k(\boldsymbol{x}_*, \boldsymbol{x}_3) & ... & k(\boldsymbol{x}_*, \boldsymbol{x}_N) \end{pmatrix} \tag{3.42}$$

The equations showed in eq. 3.40 are the core of the approach: what the GP regression does in fact is, given a specific input point x_*, providing a Normal distribution characterized by the mean value μ_{f^*} and a variance $\sigma^2_{f^*}$. Looking the new recasting in eq. 3.40, is interesting the fact that this representation of the results can be obtained from a very different point of view. From the beginning, let's define a Gaussian process as a collection of random variables, any finite number of which have a joint Gaussian distribution. This entity is used to represent the value of the function $f(\boldsymbol{x})$ at point $\boldsymbol{x}$

$$f(\boldsymbol{x}) \sim \mathcal{GP}\left(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')\right) \tag{3.43}$$

with $k(\boldsymbol{x}, \boldsymbol{x}')$ and $m(\boldsymbol{x})$ respectively covariance and mean function. Through some computation then can be shown that this covariance function is exactly what was defined before as kernel function [39], generating the same predictive distribution on f_*. This formulation links a statistical quantity, a covariance function (generalization of a covariance matrix) to kernels, a different class of mathematical object. Kernel's theory is in fact an independent field of study, quite wide and with many applications. Technically speaking a kernel is defined as a function of two variables that defines an integral operator A by the following equality

$$\psi(y) = A[\phi(x)] = \int K(x, y)\phi(x) d\mu(x) \tag{3.44}$$

when $x \in (\boldsymbol{X}, \mu)$ measure space and $\phi(x)$ defined on this space. In particular, the fact that this function is also a covariance function imposes constraints on the type of kernel, such as positive definiteness and symmetry. Moreover the definition of the term $\boldsymbol{\alpha} = \boldsymbol{k}(\boldsymbol{x}_*, \boldsymbol{X}) \left(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{Y}$, transforms the predictive mean value as

$$\mu_{f^*} = \boldsymbol{\alpha}^T \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_*) = \sum_{i=1}^{k} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}_*) \tag{3.45}$$

This equation is of paramount importance: it implies that the mean prediction for f^* can be represented in term of a linear combination of a finite number of basis function $k(\boldsymbol{x}_i, \cdot)$ This is a specific application of the so called *Mercer's Theorem* [39]. Considering in fact the eq. 3.44, is possible to define the eigenfunction of the kernel k as the function ϕ such that

$$\int k(x, y)\phi(x)d\mu(x) = \lambda\phi(x') \tag{3.46}$$

with λ corresponding eigenvalue; in general there are infinite eigenfunctions satisfying, in classical metric space (vectorial space with a measure $(\boldsymbol{X}, \mu)$), the orthonormality condition $\int \phi_i(x)\phi_j(x)d\mu(x) = \delta_{ij}$. According to the Mercer's theorem (o representation theorem), given a positive definite kernel $k(x, y) > 0$ and $\boldsymbol{\phi}$ a set of orthonormal eigenfunctions with their associated eigenvalues $\boldsymbol{\lambda} > \mathbf{0}$, the following properties hold:

$$\begin{aligned} &\{\lambda_i\}_{i=1}^{\infty} \text{ is absolute summable} \\ &k(x, y) = \sum_{i=1}^{\infty} \lambda_i\phi_i(x)\phi_i^*(y) \text{ almost everywhere} \end{aligned} \tag{3.47}$$

In case only a finite number of eigenvalues is different from zero, the kernel is said degenerate. From this theoretical framework derives a fundamental property around the Gaussian process regression. In fact, in order to numerical estimate the eigenfunctions and the eigenvalues (and so solve the problem), a set of k points is sampled (with probability $p(\boldsymbol{x})$) and used to approximate the continuous integral [40]. Since the measure of this set randomly extract is $d\boldsymbol{\mu}(\boldsymbol{x}) = p(\boldsymbol{x})d\boldsymbol{x}$, the equation becomes

$$\lambda_i\phi_i(x') = \int k(x, y)\phi_i(x)p(x)dx \approx \frac{1}{k}\sum_{i=1}^{k} k(x_k, x')\phi_i(x_k) \tag{3.48}$$

Now are evident the next steps: putting iteratively $x' = x_k$ allows to define the eigenproblem

$$\boldsymbol{K}\boldsymbol{u}_i = \lambda_i^{mat}\boldsymbol{u}_i \tag{3.49}$$

with u_i the i-th normalized eigenvector of the matrix $\boldsymbol{K}$ ($\boldsymbol{u}_i^T\boldsymbol{u}_i = 1$) and λ_i^{mat} its eigenvalue. The solution doesn't return the exact values of the eigenfunctions and eigenvalues (with respect to the original problem); what can be shown however is that, for $k \to \infty$ the estimated $\lambda_i^{mat} \to \lambda_i$ (in fact $\frac{\lambda_i^{mat}}{k}$ is an estimator of λ_i). This is one (of many possible approaches) to show heuristically that Gaussian process regression converges (in measure) to the exact value. It's interesting the fact that this approximation is quite similar to the Nyström approximation of eigenfunctions. In fact can be obtained by eq. 3.48 dividing both sides by λ_i and approximating the eigenvalue by its estimator

$$\phi_i(x') = \frac{\sqrt{k}}{\lambda_i^{mat}}\boldsymbol{k}^T\boldsymbol{u}_i \tag{3.50}$$

This approximation will be used largely in next sections for the formulation of approximate approaches.

Covariance function

The choice of the covariance function is an essential step for the use of the Gaussian process regression; in fact, between all possible kernels, there is usually a specific one more adapt than others, according to the process that must be reconstructed. Let's consider again a kernel function of the form $k(\boldsymbol{x}, \boldsymbol{x}')$ with $\boldsymbol{x} \in \mathbb{R}^D$. A class of kernels very important is the so called *stationary kernel*; the peculiarity of this type of function is the fact that depends only from the distance vector $\boldsymbol{\tau} = \boldsymbol{x} - \boldsymbol{x}'$ between the inputs, so $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{\tau})$. This feature is very convenient, since allows the application of the Bochner theorem and so the construction of the Fourier's dual of the kernel function [39], called power density $S(\boldsymbol{s})$ (and the contrary)

$$k(\tau) = \int S(\boldsymbol{s}) e^{2i\pi \boldsymbol{s}\cdot\boldsymbol{\tau}} d\boldsymbol{s} \tag{3.51}$$

$$S(\boldsymbol{s}) = \int k(\boldsymbol{\tau}) e^{-2i\pi \boldsymbol{s}\cdot\boldsymbol{\tau}} d\boldsymbol{\tau} \tag{3.52}$$

Roughly speaking this means that a stationary kernel can be represented in the Fourier's space, using as basis functions the complex exponential and in which the power density represents the coefficient (power) assigned to each basis. Note that $\boldsymbol{\sigma}^2 = k(\mathbf{0}) = \int S(\boldsymbol{s}) d\boldsymbol{s}$ so the power density must be integrable to represent a Gaussian process. In some situations can be convenient represent the kernel and the process in the Fourier's space, instead in time (or index, in case of sampled instances) domain. Another important class of kernel is the *isotropic kernel*, stationary kernel depending not from the entire vector $\boldsymbol{\tau}$ but only from its module, so $\|\boldsymbol{\tau}\| = r \rightarrow k(\boldsymbol{x}, \boldsymbol{x}') = k(r)$. Under this assumption is possible to show that the power density will depend only to the module of the complex variable $\|\boldsymbol{s}\| = s$. In fact, through a change of coordinates to spherical ones and the integration of the angular variables, the previous equation will become

$$k(r) = \frac{2\pi}{\mathbb{R}^{D/2-1}} \int_0^\infty S(s) J_{D/2-1}(2\pi r s) s^{D/2} ds \tag{3.53}$$

$$S(s) = \frac{2\pi}{s^{D/2-1}} \int_0^\infty k(r) J_{D/2-1}(2\pi r s) \mathbb{R}^{D/2} dr$$

with $J_{D/2-1}$ Bessel's function of order $D/2 - 1$. In this definition is important to note that the input dimension is the key factor: the same power density $S(s)$ can represents a different kernel, in case of different dimension D; at the same time a given kernel doesn't correspond to a unique power density function. One of the most popular and used kernel is the *Squared Exponential Kernel* (SE). It is defined as

$$k_{SE}(\boldsymbol{x}, \boldsymbol{x}') = \sigma \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^T \boldsymbol{\Lambda} (\boldsymbol{x} - \boldsymbol{x}')\right) \tag{3.54}$$

where σ and $\boldsymbol{\Lambda}$ are parameters to choice (usually named hyperparameters). In particular the first term corresponds to the *amplitude* of the kernel while the matrix inside the brackets weights the different component of the distance vector and is called *characteristics length-scale*. This kernel depends on the weighted norm of the distance vector; still can be considered as a stationary and isotropic kernel. In fact,

assuming a lenght-scale matrix positive definite, is possible again to represent it with a Cholesky decomposition as $\mathbf{\Lambda} = (\mathbf{\Lambda}^{1/2})^T(\mathbf{\Lambda}^{1/2})$, and so rewriting the argument of the exponential in the eq. 3.54 as

$$(\boldsymbol{x} - \boldsymbol{x}')^T\mathbf{\Lambda}(\boldsymbol{x} - \boldsymbol{x}') = \boldsymbol{\tau}^T(\mathbf{\Lambda}^{1/2})^T(\mathbf{\Lambda}^{1/2})\boldsymbol{\tau} = (\mathbf{\Lambda}^{1/2}\boldsymbol{\tau})^T(\mathbf{\Lambda}^{1/2}\boldsymbol{\tau}) \tag{3.55}$$

so the classic norm of a new distance vector, which directions are scaled according to $\mathbf{\Lambda}$. Usually this matrix is chosen diagonal, in order to just scale the different directions, without merging them. The corresponding power density function is

$$S_{SE}(\boldsymbol{s}) = C\exp\left(-2\boldsymbol{s}^T\mathbf{\Lambda}^{-1}\boldsymbol{s}\right) \tag{3.56}$$

with C normalization coefficient. This kernel is quite unrealistic, since modeling physical process with such smoothness is a strong assumption; despite of this, is the most widely used. A slightly different kernel, less smooth than the previous one, is the so called *Matèrn kernel*, defined as follows

$$k_{matern,\nu}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu r}}{l}\right)^{\nu} K_{\nu}\left(\frac{\sqrt{2\nu r}}{l}\right) \tag{3.57}$$

with r the norm of the difference vector (since stationary), ν and l positive parameters and K_ν modified Bessel's function and spectral density

$$S_{matern,\nu}(s) = \frac{2^D\pi^{D/2}\Gamma(\nu + D/2)(2\nu)^{\nu}}{\Gamma(\nu)l^{2\nu}}\left(\frac{2\nu}{l^2} + 4\pi^2 s^2\right)^{-(\nu+D/2)} \tag{3.58}$$

The value of ν is crucial. In fact, for $\nu \to \infty$ the Matèrn kernel converges to the Squared Exponential one; for half-integer value $\nu = p + \frac{1}{2}$, with p integer non-negative, the formula becomes very simple. Because of this, values such as $\nu = 1/2$, representing a very rough rough, while values such as $\nu = 3/2, 5/2$ are used for increasing smoothness in the representation. A non exponential class of kernels widely used is the *Rational Quadratic* kernel (RQ). This covariance can be seen as specific case of a kernel generated by the superposition of infinite SE kernel

$$k_{RQ}(r) = \int k_{SE}(r\|l)p(l)dl \tag{3.59}$$

in which length-scale hyperparameters is considered as a random variable distributed according to a particular PDF, the Gamma distribution. Below follows its mathematical representation, in which the squared inverse of the length-scale is parameterized as $\eta = l^{-2}$ to simplify the expression

$$P(\eta\|\alpha, \beta) = \eta^{\alpha-1}\exp\left(-\frac{\alpha\eta}{\beta}\right) \tag{3.60}$$

with the two parameters α, β of the Gamma function. This function has the peculiarity to be the maximum entropy probability distribution for the random variable η: this means that, assuming that nothing is known around this variable, the correct choice is a distribution that maximizes the entropy and minimizes the

amount of prior information, so the Gamma function. The derivation of the kernel follows then these steps, setting $\beta = \eta = l^{-2}$

$$k_{RQ} = \int P(\eta \| \alpha, \eta)\ k_{SE}(r \| \eta) d\eta = \tag{3.61}$$
$$\approx \int \eta^{\alpha-1} \exp\left(-\frac{\alpha\eta}{\eta}\right)\ \exp\left(-\frac{\eta \mathbb{R}^2}{2}\right) d\eta =$$
$$\left(1 + \frac{\mathbb{R}^2\eta^2}{2\alpha}\right)^{-\alpha} = \left(1 + \frac{\mathbb{R}^2}{2\alpha l^2}\right)^{-\alpha}$$

These aforementioned kernels are some of many possible choices; moreover, according to the specific task, kernels can also combined. In particular, kernels are closed with respect to the following operations:

- Given K_1, K_2 kernels, then $\rightarrow K_1 + K_2 = K_3$ is still a kernel
- Given K_1, K_2 kernels, then $\rightarrow K_1 \cdot K_2 = K_3$ is still a kernel
- Given a deterministic function $a(x)$ and a random function $f(x)$, then the kernel of this new process $g(x) = a(x)f(x)$ is $\rightarrow K_g(x, x') = a(x)K_f(x, x')a(x')$
- Given a kernel function $K(x, z)$ and the convolution map $g(x) = \int K(x, z)f(z)dz$, then the kernel of this new process g is
  $\rightarrow K_g(x, x') = \int K(x, z)K_f(z, z')K(x', z')dzdz'$
- Given K_1, K_2 kernels acting on different spaces χ_1, χ_2, then $\rightarrow K_1 \bigoplus K_2 = K_3$ and $K_1 \bigotimes K_2 = K_4$ are still kernels, acting on the product space $\chi_1 \bigotimes \chi_2$

These properties are very important. In order to correctly reconstruct a mapping thought the use of a Gaussian process, the covariance function must represent (in some sense) the shape of the signal. This means, heuristically, that given two points in the input space, the actual distance in the output space should be approximated by this kernel function. A classic example is the use of an oscillating kernel for reconstructing a sinusoidal signal. The the use of an aperiodic kernel (such as exponential) would not in represent correctly the periodicity of the original signal. In fact the distance between two periods of a periodic signal (such as sin) is 0

$$\text{distance}(\sin(x), \sin(x + T)) = 0 \tag{3.62}$$

while in a general kernel, if not explicitly considered, the distance can be different from zero

$$k_{\text{aperiodic}}(x, x + T) \neq 0 \tag{3.63}$$

This means that these points are considered by the kernel close or distant, generating an artifact covariance between them and so worsening the reconstruction. Since the task is the reconstruction of a signal not completely known a priori, usually are performed some guesses around the patterns that characterizes it. Indeed, a way to not rely only on these guesses, is to use a covariance function very general, obtained through the merging of different kernels. As said before the combination of many

kernels can express a very general behaviour, with different components (such as oscillation, damping, etc...) and/or discontinuous features. Nevertheless, also in the case of a covariance function correctly representing the trend of the signal, in order to obtain a most accurate approximation, the kernel function is still kept parametric; is such a way these extra degrees of freedom can be optimized to improve drastically the performance (an example of this is in Fig. 3.4).

Hyperparameters optimization

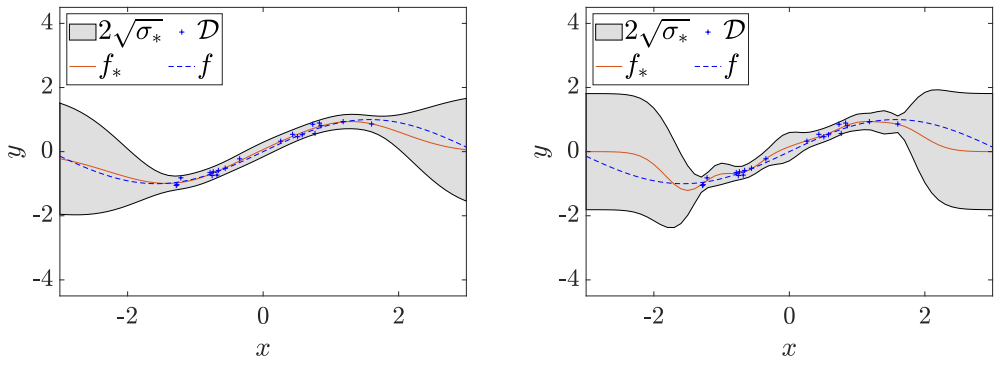

Figure 3.4. Example of GP regressions of the target function $f(x) = sin(x)$ (dashed blue). In red is plotted the mean prediction and in grey the 95% confidence interval obtained using the variance information. Furthermore, in blue and using the plus symbol are plotted the actual training points, slightly different from the true function since the noise presence. Both the regressions are obtained using a *Squared Exponential* kernel but with two different hyperparameters, one optimized (on the left) and one not optimized. Is evident how the large number of datapoints (center of the plot) reduces in both the figures the prediction variance, but at the same time the lenghtscale optimization improves (left plot) the accuracy of the prediction. image: C. E. Rasmussen & C. K. I. Williams, *Gaussian Processes for Machine Learning*, the MIT Press, 2006,ISBN 026218253X. c© 2006 Massachusetts Institute of Technology.

Differently from other classical machine learning approaches for regression, such as Neural Networks or Support Vector Machines, Gaussian process regression is a non-parametric technique, so not requiring a training phase. In fact, given a specific input, the algorithm just applies the formula for predictive distribution, providing for a given input a Gaussian distribution centered around its mean value and variance. Despite of this, as said in previous sections, the covariance function, a core quantity for representing the GP, depends from specific parameters called *hyperparameters*. These parameters are extra degrees of freedom that the user can exploit in order to improve the algorithm performance. In order to do this, a specific optimization problem is formulated, called Marginal Likelihood Maximization. The marginal likelihood is defined as the marginalization over all the possible function f of the likelihood function $P(\boldsymbol{Y}\|f, \boldsymbol{X})$, so

$$P(\boldsymbol{Y}\|\boldsymbol{X}) = \int P(\boldsymbol{Y}\|f, \boldsymbol{X})P(f\|\boldsymbol{X})df \tag{3.64}$$

Under the Gaussian process model assumption the prior around f is known to be Gaussian $P(f\|\boldsymbol{X}) = \mathcal{N}(\boldsymbol{m}(\boldsymbol{x}), \boldsymbol{K})$, obtaining eventually [39]

$$P(\boldsymbol{Y}\|\boldsymbol{X}) = \frac{1}{(2\pi)^{n_D}\sqrt{|(\boldsymbol{K}+\sigma^2\boldsymbol{I})|}}\exp\left(-\frac{1}{2}\boldsymbol{Y}^T(\boldsymbol{K}+\sigma^2\boldsymbol{I})^{-1}\boldsymbol{Y}\right) \tag{3.65}$$

Since the covariance function depends on the hyperparameters (called conventionally $\boldsymbol{\theta}_{hyper}$) also the marginal likelihood depends by them $P(\boldsymbol{Y}\|\boldsymbol{X}) = P(\boldsymbol{Y}\|\boldsymbol{X}, \boldsymbol{\theta}_{hyper})$. So, in order to find the most appropriate set of parameters, is set the following optimization problem

$$\boldsymbol{\theta}_{hyper,opt} = \underset{\boldsymbol{\theta}_{hyper}}{\arg\max}\ P(\boldsymbol{Y}\|\boldsymbol{X}, \boldsymbol{\theta}_{hyper}) \tag{3.66}$$

equivalent (as said for previous optimization problem) to

$$\boldsymbol{\theta}_{hyper,opt} = \underset{\boldsymbol{\theta}_{hyper}}{\arg\max}\ \ln(P(\boldsymbol{Y}\|\boldsymbol{X}, \boldsymbol{\theta}_{hyper})) \rightarrow \tag{3.67}$$

$$\underset{\boldsymbol{\theta}_{hyper}}{\arg\min}\ \frac{1}{2}\boldsymbol{Y}^T\left(\boldsymbol{K}(\boldsymbol{\theta}_{hyper})+\sigma^2\boldsymbol{I}\right)^{-1}\boldsymbol{Y} + \frac{1}{2}\ln\left(|\boldsymbol{K}(\boldsymbol{\theta}_{hyper})+\sigma^2\boldsymbol{I}|\right) + \frac{n_D}{2}\ln(2\pi)$$

with n_D dimension of the dataset (number of instances collected). Looking at the cost function is evident how only the first term depends actually to the observed targets $\boldsymbol{Y}$, while the second term $\ln\left(|\boldsymbol{K}(\boldsymbol{\theta}_{hyper})+\sigma^2\boldsymbol{I}|\right)$ is independent from the dataset and acts like a penalty on the kernels' complexity.

Advantages and weaknesses

Gaussian process regression presents two main benefits. First of all, is a technique that doesn't requires both training and inference steps, but just the inference procedure. This allows the use of the method online during the data collection (such as for the case of robot online model learning); secondly, it provides a measure of how much the prediction of the algorithm is reliable. In fact, for each query, the GP predicts a Normal distribution, centered around a mean value and with a specific variance. For the practical use of the method usually is just taken the mean value; because of this, in order to assure its trustworthiness, is required a distribution very peaked around it, so with a low variance (ideally a delta distribution, corresponding to a Gaussian with zero variance). Many examples of learning based controller using GP can be found in literature. In [41] a Gaussian process is used to approximate the nonlinear dynamics of a robot and propagate it inside a nonlinear MPC, interpreting the variance prediction as a probabilistic bound that can be exploited for enforcing constraints. Again in [42] a Gaussian process is used to learn the dynamics of the class of Euler-Lagrange systems, proposing a procedure to tune a PD controller according to the variance information associated with the model learning. Eventually in [43] a similar approach is employed to characterize the error bound of the GP regressor and so design a sliding mode controller able to stabilize the system. Notwithstanding GPs can be used for a large variety of controller, such as Adaptive Control [44] or Backstepping [45]. The weaknesses of this technique in some way balances the convenient features just described. Looking in fact how the

prediction is realized in eq. 3.40, is notable the fact that is required the inversion of the kernel matrix $\boldsymbol{K} \in \mathbb{R}^{n_D \times n_D}$ (3.41). Since the inversion of a square matrix of dimension n_Dxn_D scales computationally with $\mathcal{O}(n_D^3)$, as the size of the dataset increases so does time; this limits the use of this approach in an online fashion. Another more subtle weakness is the fact that the adherence of the kernel to the signal is quite fundamental. So, even tough formally a training step is not required, the hyperparameters maximization is a necessary step in order to obtain a good performance. Thus this phase cannot be realized online: in fact the computation of the inverse of the kernel matrix is still required (3.67), penalizing the use of nonlinear solvers (such as NLP, Interior Point Optimization), optimizers that usually evaluate the loss function in many different points. This issue could be partially overcome with the use of gradient based approaches; still, can be shown that the evaluation of the gradient of the marginal likelihood scales as $\mathcal{O}(n_D^2)$ [39], so hardly usable in online loops. A way to overcome these issues is the use of approximate approaches, as shown in the next section.

3.3.4 Approximate Gaussian process regression

As said in previous section, the main problem of the GPs is inversion of the kernel matrix, for both model inference and hyperparameters optimization. In order to reduce this issue, many approximated approaches were introduced. In this section just two examples are accounted: *Sparse Approaches*, in which the process is represented by a subset of m inducing variables that reduce the computational complexity of the inference and *Linearized Gaussian Process*, in which the stochastic process is approximated by another process, which inference is linear with respect to the input.

Sparse approaches

If the kernel matrix doesn't have maximum rank, so $\text{rank}(\boldsymbol{K}) \in \mathbb{R}^{n_D \times n_D} = q < n_D$ then can be decomposed as $\boldsymbol{K} = \boldsymbol{Q}\boldsymbol{Q}^T$, with $\boldsymbol{Q} \in \mathbb{R}^{n_D \times q}$. So the original problem is linked to the inversion of the matrix $(\boldsymbol{Q}^T\boldsymbol{Q}) \in \mathbb{R}^{q \times q}$ (of dimension qxq)

$$\left(\boldsymbol{Q}\boldsymbol{Q}^T + \sigma^2 \boldsymbol{I}_{\boldsymbol{n_D}}\right)^{-1} = \sigma^{-2}\boldsymbol{I}_{\boldsymbol{n_D}} - \sigma^{-2}\boldsymbol{Q}\left(\boldsymbol{Q}^T\boldsymbol{Q} + \sigma^2 \boldsymbol{I}_{\boldsymbol{q}}\right)^{-1}\boldsymbol{Q}^T \tag{3.68}$$

If the rank is maximum (so $\text{rank}(\boldsymbol{K}) = n_D$) is still possible to execute an optimal reduced ranked approximation (through SVD), selecting the first q eigenvalues

$$\boldsymbol{K} \approx \boldsymbol{U}_q \boldsymbol{\Lambda}_q \boldsymbol{U}_q^T \tag{3.69}$$

Still this decomposition scales as $\mathcal{O}(n_D^3)$, so it's not suitable for the online use. However, this idea suggests that through an approximate eigenvalues decomposition is possible to obtain a cheap approximation of the kernel matrix. Let's consider a subset I of the n_D datapoints of size $m < n_D$ and the remaining $n_D - m$ points forming the set R. Considering the points ordered such that the set I comes before the set R, the kernel matrix can be decomposed as

$$\boldsymbol{K} = \begin{pmatrix} \boldsymbol{K}_{m,m} & \boldsymbol{K}_{m,n_D-m} \\ \boldsymbol{K}_{n_D-m,m} & \boldsymbol{K}_{n_D-m,n_D-m} \end{pmatrix} \tag{3.70}$$

Similarly to the approximation of the eigenfunctions realized in eq. 3.50, in this case are approximated both eigenvectors and eigenvalues of $\boldsymbol{K}$. Firstly are computed the m eigenvalues $\{\lambda\}_{i=1}^{m}$ and eigenvectors $\{\boldsymbol{u}\}_{i=1}^{m}$ of $\boldsymbol{K}_{m,m}$. These quantities are then used to represent the generic n-th eigenvalue/eigenvector of the complete kernel matrix

$$\tilde{\lambda}_i^n = \frac{n}{m}\lambda_i^m \qquad i = 1, ...m \tag{3.71}$$

$$\tilde{\boldsymbol{u}}_i^n = \sqrt{\frac{n}{k}}\frac{1}{\lambda_i^m}\boldsymbol{K}_{n,m}\boldsymbol{u}_i^m \qquad i = 1, ...m \tag{3.72}$$

with $\tilde{\boldsymbol{u}}^n$ chosen such that $\|\tilde{\boldsymbol{u}}^n\| = 1$. The estimation of the complete kernel depends on how many approximate eigenvalues/vectors are selected; choosing the first p brings to $\tilde{K} = \sum_{i=1}^{p} \tilde{\lambda}_i^p \tilde{\boldsymbol{u}}_i^n (\tilde{\boldsymbol{u}}_i^n)^T$. In case p=m, then

$$\tilde{\boldsymbol{K}} = \boldsymbol{K}_{n_D,m}\boldsymbol{K}_{m,m}^{-1}\boldsymbol{K}_{m,n_D} \tag{3.73}$$

Computation of $\tilde{\boldsymbol{K}}$ scales as $\mathcal{O}(m^2 n_D)$, while the eigendecomposition of $\boldsymbol{K}_{m,m}$ is $\mathcal{O}(m^3)$ and the computation of $\tilde{\boldsymbol{u}}_i^n$ is $\mathcal{O}(mn)$. This is called Nysöm approximation of the kernel $\boldsymbol{K}$. Another classical method is the so called *Subset of Regressors*. This derives from the idea that the predictive mean (3.45) can be expressed as a combination of regressors $\mu_* = \sum_{i=1}^{n_D} \boldsymbol{\alpha}_i^T \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_*)$. This representation can be seen as a linear regression with a Gaussian distribution for the $\boldsymbol{\alpha} = \mathcal{N}(\boldsymbol{\alpha}\|0, \boldsymbol{K})$. In this context the approximation is quite straightforward: considering only a subset of m regressors $\boldsymbol{\alpha}$, such that

$$\mu_{*,SR} = \boldsymbol{\alpha}_m^T \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_*) = \sum_{i=1}^{m} \alpha_i^T \boldsymbol{k}(\boldsymbol{x}_i, \boldsymbol{x}_*) \tag{3.74}$$

with $\boldsymbol{\alpha}_m = \mathcal{N}(\boldsymbol{\alpha}\|0, \boldsymbol{K}_{m,m})$ Also in this method, the choice of the m points used for the approximation, usually called *active set*, is essential. Clearly is not advisable the combinatorial search of the optimal set and some heuristics must be used. Indeed, a distinct class of methods, focused on selecting the most informative set of points (that can be eventually used also for the other techniques) is called *Subset of Datapoints*. They consist in the greedy maximization at each step of a suitable cost function; an example of this is the maximization of the differential entropy score [46]. Considering the j-th point of the set R (points not considered in the active set, as said before), the points is added to the active set if is the one that (between all the points in the set R), maximizes the following function

$$\Delta_j = H(p(f_j)) - H(p(f_j)^{new}) \tag{3.75}$$

with $H(p(f_j))$ entropy calculated at the point j without including the point in the dataset and $H(p(f_j))$ with the point included. In the case of Gaussian distribution, this quantity becomes

$$\Delta_j \approx \log(1 + \frac{v_j}{\sigma^2}) \tag{3.76}$$

with v_j prediction variance. Looking at the function, is evident that is monotonic increasing with respect to vi_j, and in this case the index j of the point to include

in the active subset is the one, within all the dataset, that generates the maximum prediction variance. This procedure is iteratively executed with the adding on new points in the dataset. The goal of this approach is selecting this subset of data and eventually use it for performing a classic GP regression or use it jointly with previous methods; in both cases, this selection step scales as $\mathcal{O}(m^3)$, very convenient in the case of having milions of points and a subset I of size in the order of hundreds.

Linearized Gaussian process

Differently from previous section, this estimation acts directly on the Gaussian Process. Considering the mean value and the variance of the predictive distribution centered around $\boldsymbol{x}_*$ as nonlinear functions $\boldsymbol{x}_*$, a way to reduce inference complexity is to linearize both the mean and variance functions around a specific point through a Taylor's approximation. In such a way the new query point will consist in the different vector between the real query point and the linearization one. This can surely introduce some benefits regarding computational complexity [47], in particular for high dimensional problems. Nevertheless, the approximation so obtained is not anymore a Gaussian process and this can cause a not coherent behaviour, such as the violation of the statistical properties (e.g. negative variance) and not conservation of the distribution moments (in particular beyond the second one). Another technique, recently developed (linGP [48]), still operates a GP linearization but via a very different scheme and more important, obtaining an object that is still a Gaussian Process. Let's consider again a Gaussian process $\mathcal{GP}_f$, specified as in eq. 3.43

$$f(\boldsymbol{x}) \sim \mathcal{GP}_f\left(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')\right) \tag{3.77}$$

A fundamental assumption (for every approach based on a linearization) is the differentiability of both target and covariance functions. Let's consider a specific point $\boldsymbol{x}_* \in \mathbb{R}^{n_D}$, called *linearization point*. Since the suitable differentiability properties, the target function can be linearized around this point, getting the following result

$$f(\boldsymbol{x}_* + \boldsymbol{\Delta}_x) \approx \tilde{f}_x(\boldsymbol{\Delta}_x) = f(\boldsymbol{x}_*) + \boldsymbol{\Delta}_x^T \boldsymbol{\nabla}_x f(\boldsymbol{x}_*) \qquad \boldsymbol{\Delta}_x \in \mathbb{R}^{n_D} \tag{3.78}$$

Through a redefinition of the gradient of the function $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{\nabla}_x f(\boldsymbol{x})$ and of the input $\hat{\boldsymbol{x}} = \begin{pmatrix} 1 & \boldsymbol{\Delta}_x \end{pmatrix}$ the linearization can be rewritten as

$$\tilde{f}_x(\boldsymbol{\Delta}_x) = \begin{pmatrix} 1 \\ \boldsymbol{\Delta}_x \end{pmatrix}^T \begin{pmatrix} f(\boldsymbol{x}_*) \\ \boldsymbol{g}(\boldsymbol{x}_*) \end{pmatrix} = \hat{\boldsymbol{x}}^T \hat{\boldsymbol{f}}(\boldsymbol{x}_*) \tag{3.79}$$

Is important point out that the original GP has been approximated around $\boldsymbol{x}_*$ by a process $\tilde{\mathcal{GP}}_{f\|\boldsymbol{x}_*}$ of the linearized function $\tilde{f}_x(\boldsymbol{\Delta}_x)$, called linGP; so, in order to characterize it, must first defined the distribution of $\hat{\boldsymbol{f}}$. Since the differentiation is a linear operator, $\hat{\boldsymbol{f}}$ is Gaussian process derived from the original GP; therefore, is also a Gaussian random vector that defines a posterior distribution over the values of target function and its gradient. Similarly to what realized in [39] for the original

$\mathcal{GP}_f$, can be defined the joint distribution $P(\boldsymbol{Y}, \boldsymbol{X}, \hat{\boldsymbol{f}}, \boldsymbol{x}_*)$ as

$$P(\boldsymbol{Y}, \boldsymbol{X}, \hat{\boldsymbol{f}}, \boldsymbol{x}_*) = \mathcal{N}\left(\boldsymbol{0}, \begin{pmatrix} \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma^2\boldsymbol{I} & \boldsymbol{k}(\boldsymbol{X},\boldsymbol{x}_*) & \boldsymbol{K}^{(0,1)}(\boldsymbol{X},\boldsymbol{x}_*) \\ \boldsymbol{k}(\boldsymbol{x}_*,\boldsymbol{X}) & \boldsymbol{k}(\boldsymbol{x}_*,\boldsymbol{x}_*) & \boldsymbol{K}^{(0,1)}(\boldsymbol{x}_*,\boldsymbol{x}_*) \\ \boldsymbol{K}^{(1,0)}(\boldsymbol{x}_*,\boldsymbol{X}) & \boldsymbol{K}^{(1,0)}(\boldsymbol{x}_*,\boldsymbol{x}_*) & \boldsymbol{K}^{(1,1)}(\boldsymbol{x}_*,\boldsymbol{x}_*) \end{pmatrix}\right) \tag{3.80}$$

in which the quantities representing the covariance function of the joint distribution are defined as follows:

- $k(\cdot,\cdot)$ the covariance function (kernel) of the original process
- $\boldsymbol{K}$ the covariance matrix of the original process
- $\boldsymbol{K}^{(1,0)} = \boldsymbol{\nabla}_{\boldsymbol{x}}(k(\boldsymbol{x},\boldsymbol{x}'))$ the derivative of the kernel with respect to the first argument (Gradient)
- $\boldsymbol{K}^{(0,1)} = D_{\boldsymbol{x}'}(k(\boldsymbol{x},\boldsymbol{x}'))$ the derivative of the kernel with respect to the second argument (Jacobian)
- $\boldsymbol{K}^{(1,1)} = D_{\boldsymbol{x}'}(\boldsymbol{\nabla}_{\boldsymbol{x}}(k(\boldsymbol{x},\boldsymbol{x}')))$, the matrix of second derivatives of the kernel (Hessian)

which dimension is related to the evaluated input, e.g. $\boldsymbol{k}(\boldsymbol{x},\boldsymbol{X})$ corresponds to a vector since computed on a point and a vector (technically called broadcasting). As for classic GP, the posterior distribution of Y conditioned on the dataset, the gradient and the linearization point is [49]

$$P(\hat{\boldsymbol{f}} \| \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{x}_*) = \mathcal{N}(\hat{m}_x, \hat{V}_x) \tag{3.81}$$

with

$$\hat{\boldsymbol{m}}_x = \begin{pmatrix} \boldsymbol{k}(\boldsymbol{x}_*,\boldsymbol{X}) \\ \boldsymbol{K}^{(1,0)}(\boldsymbol{x}_*,\boldsymbol{X}) \end{pmatrix} \left(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma^2\boldsymbol{I}\right)^{-1} \boldsymbol{Y} \tag{3.82}$$

$$\hat{\boldsymbol{V}}_x = \begin{pmatrix} k(\boldsymbol{x}_*,\boldsymbol{x}_*) & \boldsymbol{K}^{(0,1)}(\boldsymbol{x}_*,\boldsymbol{x}_*) \\ \boldsymbol{K}^{(1,0)}(\boldsymbol{x}_*,\boldsymbol{x}_*) & \boldsymbol{K}^{(1,1)}(\boldsymbol{x}_*,\boldsymbol{x}_*) \end{pmatrix} - \begin{pmatrix} \boldsymbol{k}(\boldsymbol{x}_*,\boldsymbol{X}) \\ \boldsymbol{K}^{(1,0)}(\boldsymbol{x}_*,\boldsymbol{X}) \end{pmatrix} \left(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma^2\boldsymbol{I}\right)^{-1} \begin{pmatrix} \boldsymbol{k}(\boldsymbol{X},\boldsymbol{x}_*) & \boldsymbol{K}^{(0,1)}(\boldsymbol{X},\boldsymbol{x}_*) \end{pmatrix}$$

From this, according to eq. 3.79

$$\tilde{f}_x(\boldsymbol{\Delta}_x) = \mathcal{N}(\hat{\boldsymbol{x}}^T\hat{\boldsymbol{m}}_x, \hat{\boldsymbol{x}}^T\hat{\boldsymbol{V}}_x\hat{\boldsymbol{x}}) \tag{3.83}$$

The linGP $\tilde{\mathcal{GP}}_{f\|\boldsymbol{x}_*}$ just obtained is a Gaussian Process, so preserving the statistical properties of a sthocastic process. In particular the predictive distribution has mean value $\mu_{f\|\boldsymbol{x}_*} = \hat{\boldsymbol{x}}^T\hat{\boldsymbol{m}}_x$ and variance $\tilde{\sigma}^2 = \hat{\boldsymbol{x}}^T\hat{\boldsymbol{V}}_x\hat{\boldsymbol{x}}$. Looking eq. 3.83 is interesting the fact that the mean value obtain through just the simple linearization of the equations in 3.40 is the same of linGP prediction; indeed, the variance is quite different and more accurate in the latter case. The use of this linGP is very convenient for many reasons. In fact, differently from sparse approaches, the inversion of the

complete kernel matrix is still realized, but corresponds to just an initialization, since coefficient vector $\hat{\boldsymbol{m}}_x$ and variance matrix $\hat{\boldsymbol{V}}_x$ are then fixed. Undeniably, as classic linearization based approaches, the validity of the method depends from the distance of the new input w.r.t. the linearization point $\boldsymbol{x}_*$. Even so, through the combination of a sparse approach for points selection (such as *Subset of Datapoints*) with this technique can reduce drastically the time complexity of the GP regression.

Chapter 4

Online learning with feedback linearization control of robots

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, during robot interactions with the environment (for example when implementing strategies for the sensorless detection, isolation and reaction to unexpected collisions [50]), or when regulating force or imposing a desired impedance control at the contact [51] is required. As stated in Ch. 3 many methods were employed in order to retrieve a reliable estimation of the dynamic model, from the classic model identification to more modern machine learning based approaches. In this chapter is proposed a mechanism to reconstruct the dynamic model uncertainties and parameter variations by means of an online algorithm based on GP regression (Sec. 3.3.3). Having an *a-priori* estimation of the dynamic model of a robot, the method allows the improvement of the model by exploiting only the joint position measurements, without the need for any joint torque data. This is very convenient: torque measurements in fact are usually affected by a high level of noise and the use of just encoders provides a more reliable estimation of the dynamic mismatch. According to what said before around different classed of model learning in robotics, this technique belongs to the family of inverse model learning(Sec. 3.3.1). A classic in this field is [52], in which the authors describe a way to learn the FL missing terms through the exploit of torque measurements dataset. Nevertheless, the main difference with the proposed method consists in the clear definition of the missing terms, that allows to connect the torque mismatch to the model mismatch and then build a learning procedure using only encoders output, without joint torque measurements, in general very noisy. The scheme utilizes also a FL correction in cascade of a MPC, which task consists in the high level planning. This is quite different with respect classic direct learning approaches that applies preview controllers, such as [53]. In these cases the regressor is used to correct the nominal transition model, while in the proposed method the MPC works for shaping an optimal behaviour for the double integrator system, so the already correctly feedback linearized system.

4.2 Problem formulation

The problem formulation will be briefly reviewed, but for full details the reader can refer to Ch. 2, in particular Sec. 2.1.2. For a robot fully actuated with n degree of freedom and n actuators, its dynamics can be written as

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

with the same notation expressed in previous chapters. Supposing that the robot is fully actuated, it is possible to design a FL controller that cancels the nominal nonlinear dynamics of the model (Sec. 2.1.2) as

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

where $\boldsymbol{u}$ represents the desired joint accelerations vector. In principle, assuming a perfect knowledge of the system dynamic, so $\boldsymbol{M} = \hat{\boldsymbol{M}},\ \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \hat{\boldsymbol{c}}(\boldsymbol{q}, \dot{\boldsymbol{q}}),\ \boldsymbol{g}(\boldsymbol{q}) = \hat{\boldsymbol{g}}(\boldsymbol{q})$, with the application of the nominal FL controller the closed loop dynamics would result in a linear system of double integrator

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

In the presence of a mismatch between the nominal and real parameters the resultant dynamics will be instead a perturbed double integrator

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}\hat{\boldsymbol{M}}\boldsymbol{u} - \boldsymbol{M}^{-1}\Delta\boldsymbol{n} = \boldsymbol{u} + \boldsymbol{\delta}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \tag{4.4}$$

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. The main idea of the approach is the online retrieve and compensation of $\boldsymbol{\delta}$, to directly eliminate the remaining unknown dynamics.

4.3 The proposed approach

The proposed learning based methods is able to execute an accurate trajectory tracking through the online adjustment of a model based nominal control architecture. In details, the nominal control scheme is composed by an high level Model Predictive Controller, which works assuming a double integrator transition model, followed by a feedback linearization module in cascade. Externally from all these blocks, a learning based correction is employed in order to rectify the FL step; this module continuously updates an estimation $\boldsymbol{\varepsilon}$ of the perturbation term $\boldsymbol{\delta}$, that is going to be used to compensate the unwanted term and obtain the desired double integrator dynamics. To avoid the use of torque measurements during the learning process, an estimate of the system acceleration is realized using the so called *Controllability Gramian*.

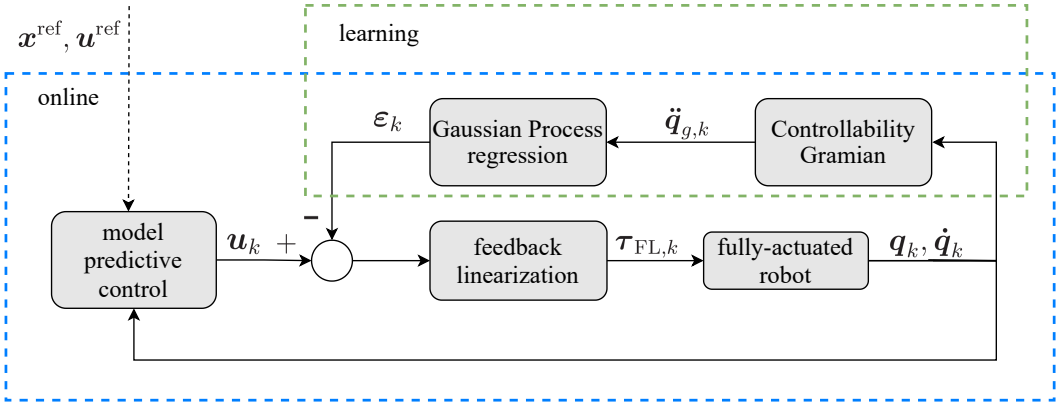

Figure 4.1. Block diagram of the proposed algorithm. The scheme is the following: the MPC (considering as transition model the double integrator) provides an high level reference acceleration to the nominal FL block. This reference is modified through the use of the GP prediction, which provides the correction in order to actually employ the feedback linearization. Eventually the FL torque is provided to the robot motors; this is repeated in loop at each control step.

4.3.1 Control architecture

The control scheme is composed by three main sections: the high level model predictive controller, the feedback linearizer module and the Gaussian process based corrector (Fig. 4.1). The main task of the architecture is the accurate tracking of a reference trajectory even in the presence of only a partially correct robot dynamic model. The first step consists in the MPC evaluation, in order to provide a suitable reference acceleration for the FL block. As said before, this controller uses as transition model the double integrator, so assuming a perfect linearization. The use of the MPC allows the avoid of unfeasible control actions or unfeasible reached states (i.e., out of known mechanical ranges) while providing an optimal and smooth joint reference acceleration $\boldsymbol{u}_k$. In acts as an high level controller; still, the framework is not hinged with this particular choice and many other control laws could be employed, such as PD or LQR.

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. Considering the state $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})^T$, for the j-th degree of freedom the continuous equations of the double integrator are the following

$$\begin{aligned} \dot{\boldsymbol{x}}^j(t) &= \boldsymbol{A}\boldsymbol{x}^j(t) + \boldsymbol{B}u^j(t) \\ \boldsymbol{y}^j(t) &= \boldsymbol{C}\boldsymbol{x}^j(t) \end{aligned} \tag{4.5}$$

where $\boldsymbol{A} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$, $\boldsymbol{B} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, $\boldsymbol{C} = \begin{pmatrix} 1 & 0 \end{pmatrix}$ (assuming the state measurable) and u^j the j-th input. Indeed, in order to numerically formulate the problem, the continuous equations are discretized using a First Order Hold (FOH) approximation, so considering the input constant between two output's sample. This discretization is very accurate if the sampling period is small enough; the resulting equations are

the following

$$\begin{aligned} \boldsymbol{x}_{k+1}^{j} &= \begin{pmatrix} 1 & T_s \\ 0 & 1 \end{pmatrix} \boldsymbol{x}_{k}^{j} + \begin{pmatrix} 0.5\ T_s^2 \\ T_s \end{pmatrix} u_{k}^{j} \\ \boldsymbol{y}_{k}^{j} &= \begin{pmatrix} 1 & 0 \end{pmatrix} \boldsymbol{x}_{k}^{j} \end{aligned} \tag{4.6}$$

with Ts sampling period; this formulation can be easily inserted as transition model of a linear MPC. Suppose now to have a precomputed reference trajectory ($\boldsymbol{x}^{\text{ref}}(t)$, $\boldsymbol{u}^{\text{ref}}(t)$); its discretization, with a sampling time Ts (the same used for the system), generates the finite set of trajectory points ($\boldsymbol{x}_k^{\text{ref}}$, $\boldsymbol{u}_k^{\text{ref}}$) for $k = 0, .., T/T_s$. Because of the fact that both dynamics and constraints are linear, also the MPC at time t_k can be formulated as linear, in the following form

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

subject to

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

with $\boldsymbol{x}_0 = \boldsymbol{x}_k$ the initial state of the robot at time t_k and $\boldsymbol{g}(\cdot)$ and $\boldsymbol{h}(\cdot)$ the desired state and input constraints. As said in Sec. 2.2, the cost is composed by two components, a stage and a terminal cost: the first one penalizes both the integrated error with respect to the reference trajectory and the smoothness of the control input $\boldsymbol{u}$, while the second one weights the distance between the final state $\boldsymbol{x}_N$ and the corresponding trajectory point $\boldsymbol{x}_N^{ref}$

$$\begin{aligned} J_s(\boldsymbol{x}_i, \boldsymbol{u}_i) &= (\boldsymbol{x}_i^{\text{ref}} - \boldsymbol{x}_i)^T \boldsymbol{Q} (\boldsymbol{x}_i^{\text{ref}} - \boldsymbol{x}_i) + (\boldsymbol{u}_i - \boldsymbol{u}_{i-1})^T \boldsymbol{R} (\boldsymbol{u}_i - \boldsymbol{u}_{i-1}), \\ J_N(\boldsymbol{x}_N) &= (\boldsymbol{x}_N^{\text{ref}} - \boldsymbol{x}_N)^T \boldsymbol{Q}_N (\boldsymbol{x}_N^{\text{ref}} - \boldsymbol{x}_N) \end{aligned} \tag{4.7}$$

where $\boldsymbol{Q}$, $\boldsymbol{Q}_N$, and $\boldsymbol{R}$ are 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 the 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. Since the precise feedback linearization is obtained only after that the dynamic mismatch is correctly estimated and compensated, the constraints satisfaction is not guaranteed at the beginning of the learning process.

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 has become negligible, the optimal control input from the MPC will assure constraint satisfaction for the real robot as well.

Given the optimal input sequence $\boldsymbol{U}^{opt} = \{\boldsymbol{u}_0^{opt}, \boldsymbol{u}_1^{opt}, ..., \boldsymbol{u}_N^{opt}\}$, the following step consists in using the solution $\boldsymbol{u}_0^{opt}$ as reference acceleration for the FL block; although, since the presence of the dynamic mismatch, is eventually subtracted a compensating term $\boldsymbol{\varepsilon}$, such as the actual reference acceleration will consist in

$$\boldsymbol{u}_{ref} = \boldsymbol{u}_0^{opt} - \boldsymbol{\varepsilon} \tag{4.8}$$

After that the nominal FL joint torque is evaluated, using the current state and this new updated reference acceleration $\boldsymbol{u}_{ref}$, and commanded to the motors. The procedure is then repeated at each control step.

One of the novelty of all the framework is the online learning and compensation of the dynamic mismatch. These disturbances are firstly approximated, using a regressor $\boldsymbol{\varepsilon}$ fitted to a specific dataset, and then online canceled. The use of a peculiar regression technique in this context is irrelevant; in the numerical simulations was employed Gaussian process regression, for its convenient properties (for more details Sec. 3.3.3), but any other methods (such as Neural Network, Least Square, etc...) could have been exploited. The main innovation is instead in the very specific unique dataset collection, analyzed in detail in the next sections.

4.3.2 Dataset collection procedure

Considering a double integrator system being at the initial state $\boldsymbol{x}_0$ at time t_0 and a specific input (and so acceleration), is possible to evaluate what state will be reached in a certain time interval (both for continuous and discrete version of the same system). In the same way, the state's evolution of a feedback linearized robot system can be evaluated. In particular, for a discrete time double integrator, the nominal FL torque steers, in an integration step, the system from the state $\boldsymbol{x}_k$ to the expected one $\boldsymbol{x}_{k+1}^{exp}$ (depending from the reference acceleration $\boldsymbol{u}_k$)

$$\hat{\boldsymbol{\tau}}_{\mathrm{FL,k}}(\boldsymbol{x}_k, \boldsymbol{u}_k) : \boldsymbol{x}_k \rightarrow \boldsymbol{x}_{k+1}^{exp}$$

Considering now a system in which the nominal FL is approximate (so incorrect): from the same state and reference acceleration as before the system will evolve to a new state $\boldsymbol{x}_{k+1}^{real}$, different from $\boldsymbol{x}_{k+1}^{exp}$. At this point, the torque assigned can be seen as the nominal (incorrect) FL law with the designed reference acceleration $\boldsymbol{u}_k$ or equivalently the correct FL law with a different reference acceleration. This equivalence can be claimed only for feedback linearization, since its coincidence with the robot model. Mathematically speaking this means that

$$\boldsymbol{M}_k\ddot{\boldsymbol{q}}_k + \boldsymbol{c}_k + \boldsymbol{g}_k = \boldsymbol{\tau}_{\mathrm{FL,k}}(\boldsymbol{x}_k, \ddot{\boldsymbol{q}}_k) = \hat{\boldsymbol{\tau}}_{\mathrm{FL,k}}(\boldsymbol{x}_k, \boldsymbol{u}_k) = \hat{\boldsymbol{M}}_k\boldsymbol{u}_k + \hat{\boldsymbol{c}}_k + \hat{\boldsymbol{g}}_k \tag{4.9}$$

Not surprisingly the term $\ddot{\boldsymbol{q}}_k$ is the actual acceleration realized from the system at time t_k with $\hat{\boldsymbol{\tau}}_{\mathrm{FL,k}}$. At this point the perturbation term $\boldsymbol{\delta}$ can be computed according to eq. 4.4

$$\ddot{\boldsymbol{q}}_k - \boldsymbol{u}_k = -\hat{\boldsymbol{M}}^{-1}(\Delta\boldsymbol{M}\ddot{\boldsymbol{q}}_k + \Delta\boldsymbol{n}). \tag{4.10}$$

In the equation 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 $\boldsymbol{x}_{k+1}^{real}$. In order to do this is used a mathematical tool called *Controllability Gramian*(CG) (explained in detail in next section), which allows to calculate approximately (almost exact under certain conditions) the joint accelerations realized by the system for the state transfer from $\boldsymbol{x}_k \to \boldsymbol{x}_{k+1}^{real}$.

Assuming to be able to evaluate $\ddot{\boldsymbol{q}}_k$, at each control step t_k the instances of $\boldsymbol{\delta}_k = \boldsymbol{\delta}_k(\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \ddot{\boldsymbol{q}}_k)$ are collected and used to incrementally build a dataset $\mathcal{D} = \{(\boldsymbol{X}_i, \boldsymbol{Y}_i) \,|i = 1, \ldots, n_d\}$, where n_d is the number of elements. For each sample the datapoint is composed by $\boldsymbol{X}_k$, the input from which depends the perturbation and its actual instance $\boldsymbol{Y}_k$

$$\boldsymbol{X}_k = (\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \ddot{\boldsymbol{q}}_k); \quad \boldsymbol{Y}_k = \ddot{\boldsymbol{q}}_k - \boldsymbol{u}_k$$

Is important to point out that, differently w.r.t. to other classical learning based methods for FL correction, no torque information is required.

The peculiarity of this learning scheme is the fact that there isn't a separation between the phases of dataset collection and FL correction. This means that during the task execution the learning module collects datapoints and at the same time provides an estimate of the required torque correction $\boldsymbol{\varepsilon}_k(\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \boldsymbol{u}_k)$. This is very convenient, since allows inline the trajectory tracking (without requiring a training phase); although, with respect to eq. 4.3.2, the datapoint is modified in order to remove this correction, that otherwise would distort the dataset

$$\boldsymbol{X}_k^{online} = (\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \ddot{\boldsymbol{q}}_k); \quad \boldsymbol{Y}_k^{online} = \ddot{\boldsymbol{q}}_k - \boldsymbol{u}_k - \boldsymbol{\varepsilon}_k$$

As just said, at each control step t_k the regressor is queried in order to provide an estimate of the extra term $\boldsymbol{\delta}_k$

$$\boldsymbol{\varepsilon}_k \to \boldsymbol{\delta}_k$$

In such a way, this extra acceleration term is subtracted to cancel the the $\boldsymbol{\delta}$ term, obtaining eventually

$$\hat{\boldsymbol{\tau}}_{Fl,k}(\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \boldsymbol{u}_k - \boldsymbol{\varepsilon}_k) = \hat{\boldsymbol{M}}(\boldsymbol{u}_k - \boldsymbol{\varepsilon}_k) + \boldsymbol{c}_k + \boldsymbol{g}_k \tag{4.11}$$

Clearly this machinery is exact only in the case of $\boldsymbol{\delta} = \boldsymbol{\varepsilon}$. Unfortunately, due to the prediction error, there is always a small error, hopefully reducing over time. Notice that in the published version of this work [12] was used a modified of eq.4.11, 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.

4.3.3 Controllability Gramian

As mentioned before, a core step of the dataset collection is the reconstruction of the realized joint accelerations $\ddot{\boldsymbol{q}}_k$. The explanation starts considering again the discretized double integrator system (4.6).

This system is controllable since

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

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

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

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

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

without using any numerical differentiation tools.

This approach can be applied according to the equivalence presented in 4.9. In fact the unexpected motion can be interpreted as if the was correctly feedback linearized via the nominal control law but driven by another unknown acceleration reference, that a posteriori can be evaluated. Following this interpretation, it is always possible to use the CG on the 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. In simulation, using realistic robot dynamic parameters the difference between the real instantaneous acceleration and the posterior obtained with the CG is on average 10^{-11} rad/s^2 (so overlapping).

4.4 Simulation results

In this section are reported the results applying the proposed method on a simulated KUKA LBR iiwa 7 R800 manipulator performing a trajectory tracking task (Fig. 4.2). The tracking performance of the nominal FL controller with and without the learning correction were tested on an altered version of the original robot, modified to include a perturbation on the nominal dynamic parameters (the ones reported in [54, 55]) of around 20 %. In detail, the masses and CoM positions were modified (alternatively increased or shrinked) and the friction term removed from the nominal model. The system was sampled and controlled with a $Ts \approx 1ms$, while the high-level MPC and the dataset acquisition run at 200 Hz. The cost matrices in eq. 4.7 are diagonal, and equal to $\boldsymbol{R} = \boldsymbol{I}_n$, while $\boldsymbol{Q} = \boldsymbol{Q}_N = 2\boldsymbol{I}_{2n}$. The trajectory belongs to the joint space and consists for the first six joints to a sinusoidal path of T=7s, while the last one a fixed position. In Fig. 4.3 is reported the absolute value of the error for the different joints during the tracking: red line for the nominal FL while blue line for the FL

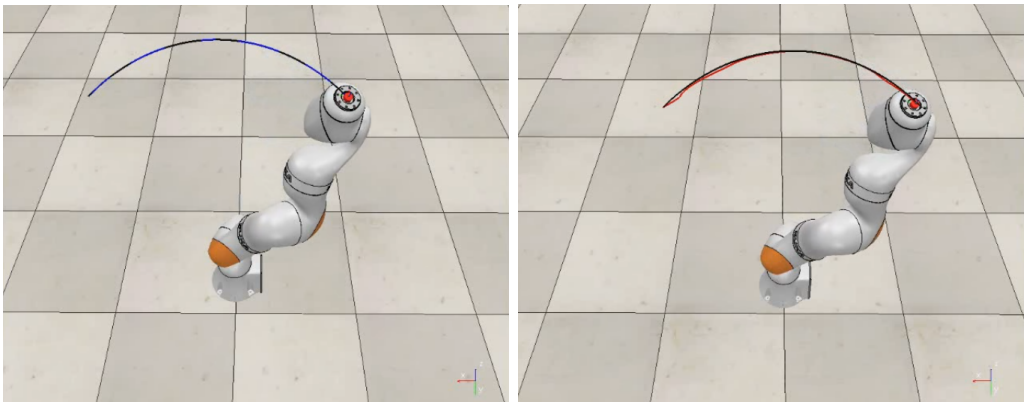

Figure 4.2. A KUKA LBR iiwa realizing a trajectory tracking task with the nominal controller (right, red) and with the proposed learning method (left, blue). A video of the simulation can be found at the following link `https://www.youtube.com/watch?v=PfEt2G9MpHU&ab_channel=GiulioTurrisi`.

+ GP correction control law. Is evident how the learning improves the tracking performance of the controller; from the starting of the motion the error is reduced in almost all the joints. Indeed, low error in the joint space has repercussion also in the Cartesian space. In fact, since the nonlinearity of the kinematics, the end effector trajectory can drastically change also in case of small angular deviations from the reference path. Fig. 4.4 reports the Cartesian error at the end effector level, with same colours as previous plot. Even in this case, the use of the learning routines improves the quality of the tracking controller. Therefore in Fig. 4.5 is plotted a comparison between the learned regressor $\boldsymbol{\varepsilon}$ and the true mismatch $\boldsymbol{\delta}$, divided for each joint. Is evident how, above all for first joints, the Gaussian process is able to reconstruct the different signals accurately. Eventually in Fig. 4.6 are showed the reference accelerations computed by the MPC (red line) and the actual ones realized by the robot (blue line). In the case of a perfect Feedback Linearization, these signals should overlap; this almost happens when the GP correction is introduced (on the left for joint 1 and 6) while in the nominal case this is not clearly happening (on the right, same joints).

4.5 Chapter summary

In the chapter has been presented a new approach for the online learning based correction of a nominal feedback linearization control for a robot manipulator with inaccurate dynamics. The scheme is composed by an high level MPC, which delivers the reference acceleration to the nominal FL module and by a learning block, which provides the correction in order to effectively realize that linearization while updating online the knowledge around the unknown dynamics. The main novelty of the work is the dataset collection procedure that, through the use of the Controllability Gramian, avoids the use of torque measurements in the reconstruction phase, reducing drastically the noise acting in the approximation. The proposed method was tested on a simulated robot manipulator, showing its high performance both in the tracking task and in the extra term reconstruction.

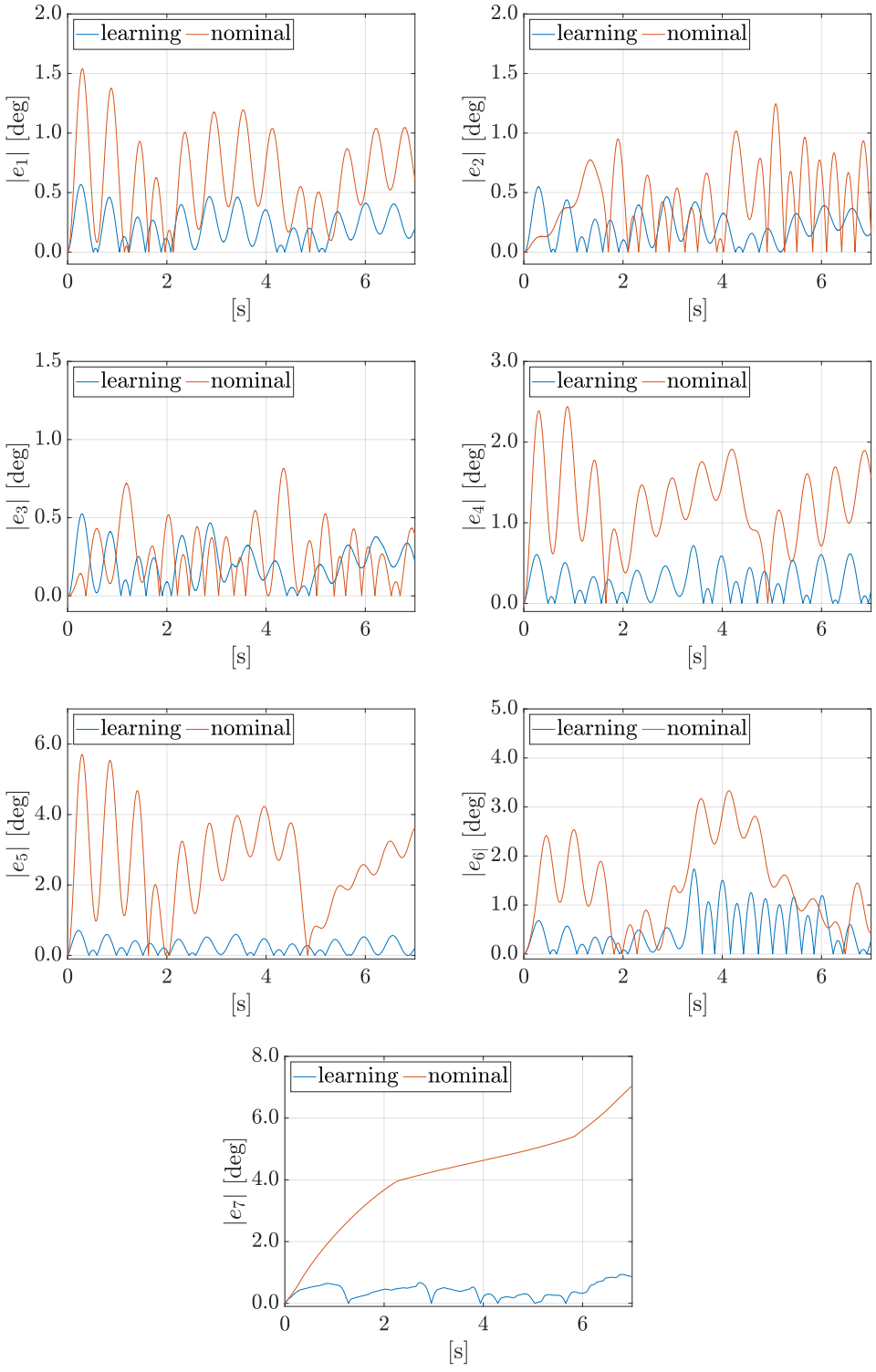

Figure 4.3. The plot shows the absolute value of the joints position errors with respect to the reference trajectory: in red represents the error obtained just the nominal FL controller while in blue is shown the one obtained using the nominal FL + GP correction. Is quite evident how the learning based approach improves the performance, in all the joints.

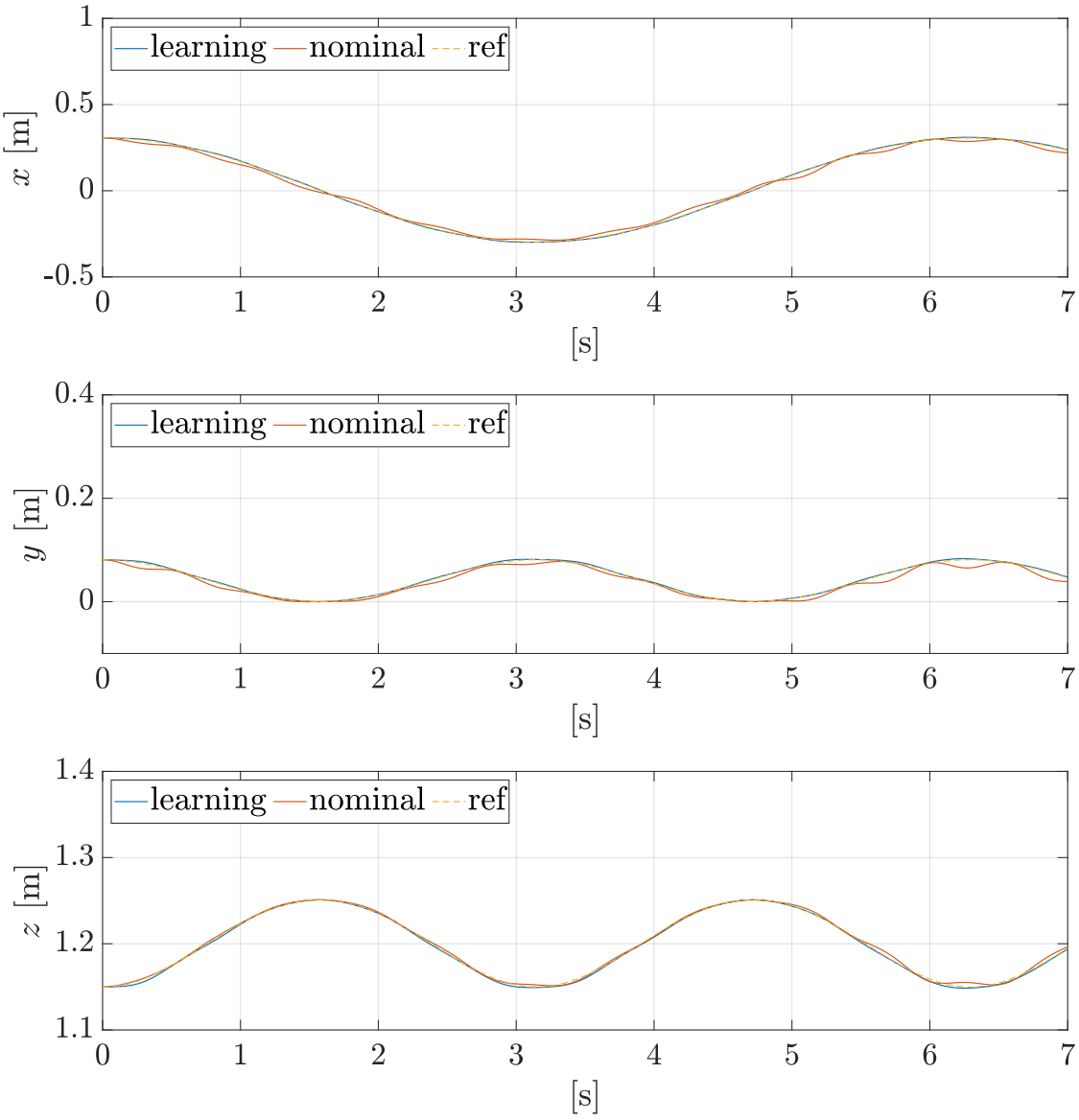

Figure 4.4. In the plot are shown the different Cartesian (x, y, z) components of the end effector trajectory realized by the system using the different controllers: nominal FL (red line), nominal FL + GP correction (blue line) and the reference trajectory the dashed yellow line. Is evident how the red and yellow line mostly overlap while the blue one slightly deviates from them.

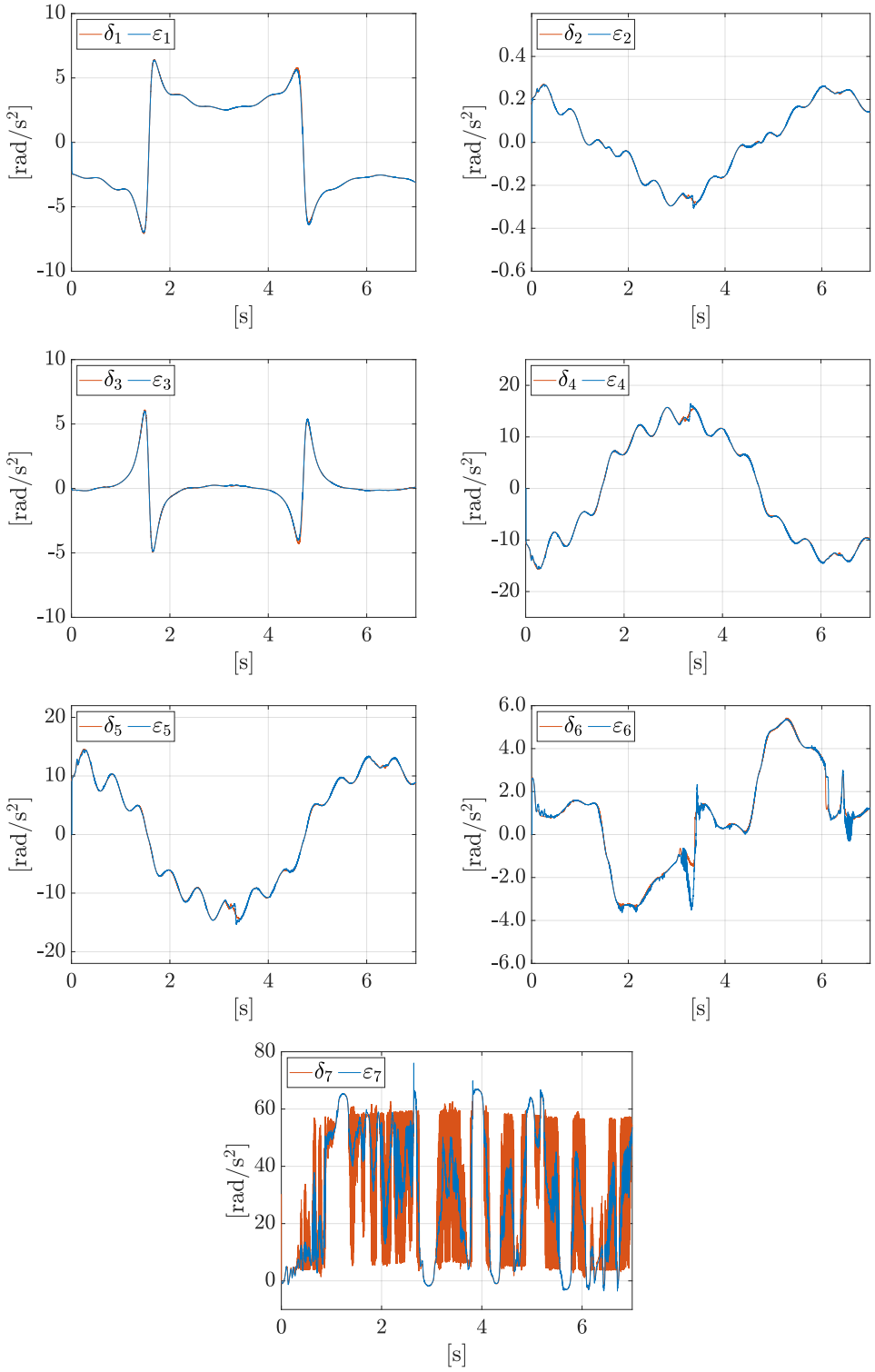

Figure 4.5. Comparison between the GP predictions $\boldsymbol{\varepsilon}$ and the mismatch signals $\boldsymbol{\delta}$ for each joint. The last joint, given its mass and inertia, is the more susceptible to the acceleration mismatch; still the GP filters the dynamic noise (oscillations), improving drastically the performance.

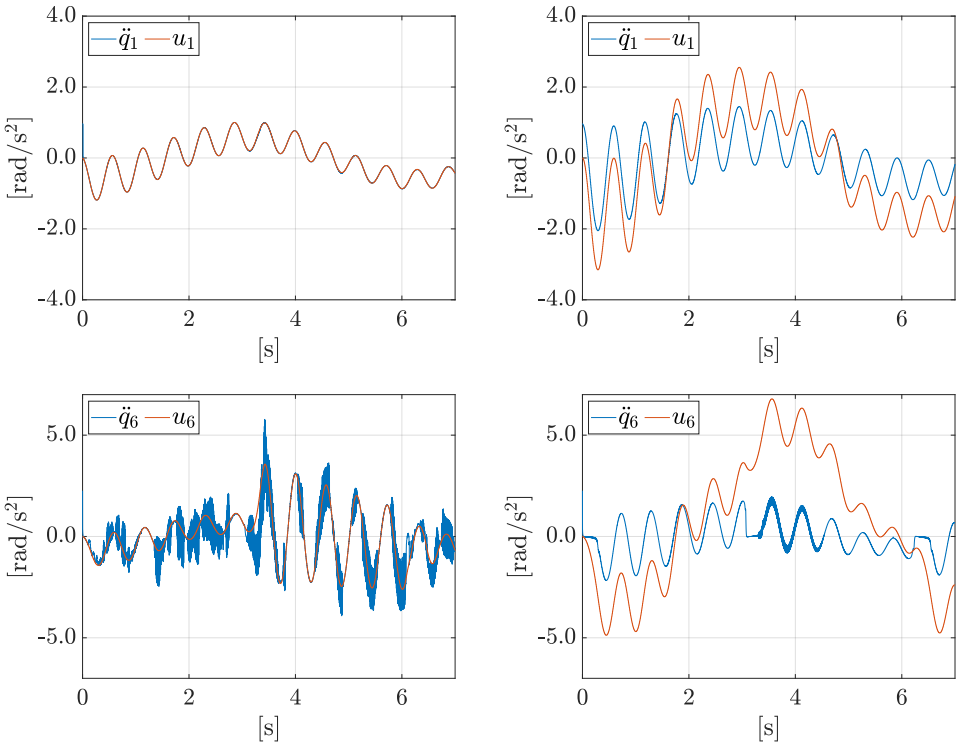

Figure 4.6. Comparison between the commanded accelerations $\boldsymbol{u}$ and the actual accelerations $\ddot{\boldsymbol{q}}$ realized by the robot for the joints 1 and 6: on the left the system is controlled by the nominal FL + GP correction, while on the right is used just the nominal FL. Is evident how the learning based correction improves drastically the feedback linearization.

Chapter 5

Online learning for planning and control of underactuated robots

5.1 Motivation and contribution

Underactuated systems nowadays are widespread in robotics, generating an independent large field of research [17]. As specified in detail in Sec. 2.1.3, roughly speaking underactuation consists in having less inputs than system's degree of freedom. This condition can be very convenient, since fews actuations means less motors; motors are quite heavy and require power supply, so their removal allows both the reduction of the energy consumption and the system's weight. The major part of modern robotics platforms present a certain degree of underactuation, including elastic/soft robots, aerial vehicles and legged robots. An adverse effect of the underactuation is the introduction of a limit in the system motion. In fact the dynamics of the passive degrees of freedom (2.17) acts as a constraints on the instantaneous accelerations achievable, inhibiting some directions and allowing others. In such a way not all the trajectories are allowed, controllability and reachability are not globally assured and trajectory planning, that for fully actuated systems is a trivial problem, becomes a demanding task. Moreover, as analyzed in detail in Sec. 2.1.3, linearization via feedback cannot be directly exerted on the entire state, but only to a specific subset of the generalized coordinates, generating at the same time nonlinear effects on the remaining DOFs. Despite of this, several model based approaches have been proposed for controlling underactuated robots, each one suited for the specific mechanical structure. The object of the interest of this manuscript is a particular class of underactuated system, consisting in robot manipulators having *passive joints* [16], links not directly actuated but only driven by the dynamic coupling within robot's components. In this category, the most simple but also the most hardest system to control is the 2R robot manipulators on the vertical plane, in which alternatively is removed the actuation from the first (called Acrobot) and second (called Pendubot) joint. Given their specific structure, both the systems present unforced equilibria states (both stable and unstable); these configurations are named according to the position of both the links: up-up to indicate the links pointing upward, down-up when the first points down and the second upward and so on. Since the presence of the underactuation constraints, the transfer between two equilibria cannot be

realized easily and has become a classic benchmark and goal that every control scheme has to achieve. In particular, the transfer from the stable down-down to the up-up/down-up unstable equilibria is called *swing-up* maneuver. This task has been solved in many ways; historically, the first controllers were based on the use of the Partial Feedback Linearization (already tackled in Sec. 2.1.3) in combination with energy based controllers [56, 57]. After that, swing up has been achieved using passivity based controllers [58, 59], orbit stabilization [60], reinforcement learning [61] and many other approaches. An important class of techniques, baseline for the proposed method, are the ones consisting in an initial optimal planning phase followed by the online tracking of the planned trajectory [17]; a very representative work in this context is [62]. Must be noticed that the general problem is usually formulated as the transfer from the starting state to the basin of attraction of the final equilibrium. Indeed, when the system is around the equilibrium, the balancing is realized through the use of a Linear Quadratic Regulator, designed according to the different system. All of these methods are tailored for the specific platform and, more important, require an accurate knowledge of the robot dynamic. In order to control this class of systems also in the presence of an uncertain dynamics, learning based approaches have been largely employed. Many methods, having some similarities with the proposed method or accounting the same task, exists in literature, both in the fields of model and control learning. In [63, 64] and [65], model-based RL techniques are used to generate robot control policies in a data-efficient way; nevertheless, these algorithms doesn't assure constraints satisfaction for the specific task. In [66, 67] is used an iterative learning approach to realize a trajectory tracking task: at each iteration an optimization step is realized to incrementally improve the feedforward input. In [68], a learning scheme is instead proposed to make a Futura's pendulum realized a trajectory tracking; despite of this, the actual task realized in testing is a simple balancing, in which the reference trajectory of the active joints is kept fixed. This limits grandly the method, not allowing the stabilization in the large. In [69] a model free RL approach has been proposed for the swing-up of an Acrobot. Despite of this effectiveness, the method requires a large training phase and yet suffers from the reality gap and generalization problem, a common issue for control learning methods. The method here proposed is grounded on the one described in Ch. 4. Notwithstanding, it accounts a slightly different problem: the realization, by an underactuated robot, of a state transfer maneuver in the presence of dynamic model uncertainties. The main idea of the approach is the modification of the classic offline trajectory planning and online tracking control framework, to introduce the acquired experiences and enhance both these phases. After the partition of the system in passive and active degrees of freedom (shown in Sec. 2.1.3), the tracking performance of the actuated joints could be improved online just exploiting the aforementioned technique (Ch. 4); nevertheless, this is not sufficient to realize the swing-up maneuver. The planning phase in fact is realized using the nominal model, generating so a trajectory that is not dynamically feasible for the robot (since the planned zero dynamics diverges from the actual one). To improve the transition model of the planner and so realize a feasible trajectory, these experiences need to be provided also during the planning phase. In such a way the adjusted planner and controller try iteratively the maneuver until the task achievement; in case of fail, the data are still kept and used for updating the framework. Over the iterations are

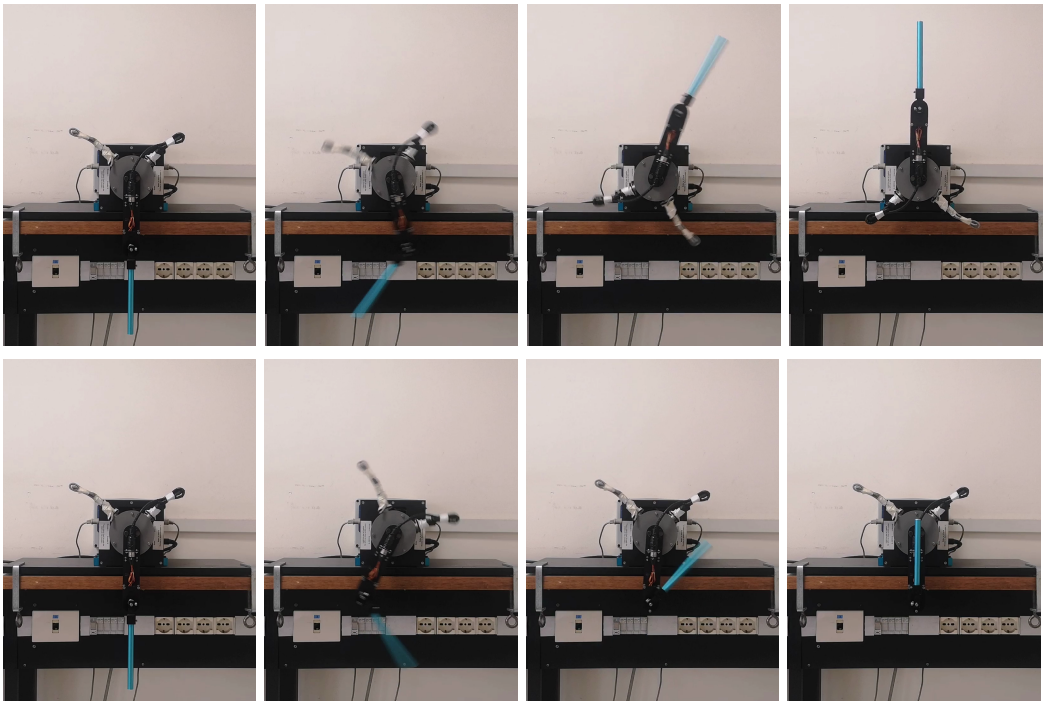

Figure 5.1. The Pendubot performing two different swing-up maneuvers using the proposed method: the targets are the up-up equilibrium (first row) and the down-up equilibrium (second row).

obtained more and more feasible trajectories while reducing the trajectory tracking error. The framework has many advantages:

- it is not hinged to a particular underactuated platform (differently from other methods)
- can be used for any state transfer maneuver
- requires very few iterations also with large uncertainties in model dynamics
- constraints can be inserted in the planning phase

5.2 Problem formulation

For a robot with n degrees of freedom (dof) and $m < n$ actuators, the dynamics can be expressed (according to Sec. 2.1.3 and using the same notation) as

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

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

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. For this class of system the linearization via feedback can be obtained alternatively on the active or passive joints, performing the so called *Collocated/Not Collocated* Partial

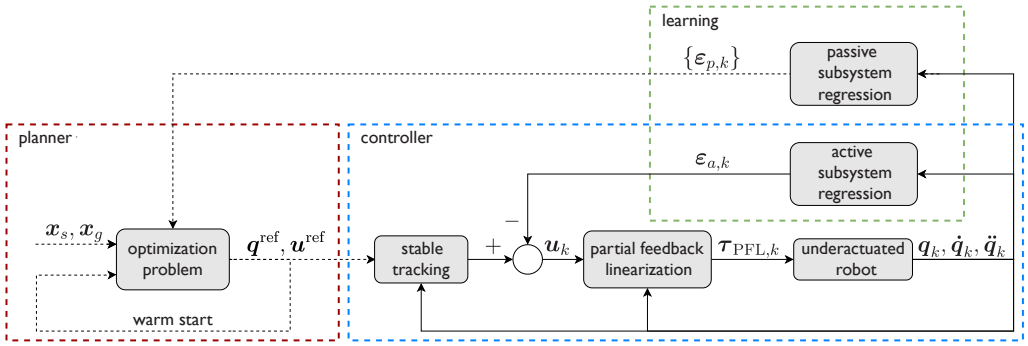

Figure 5.2. Block diagram of the generic iteration of the proposed algorithm. Solid signal lines represent data that are used at each time step, whereas dashed lines are data transferred at the end of the iteration.

Feedback Linearization (PFL). For the partitioned system (eq. 2.17) the nominal collocated PFL consists in

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

in which $\boldsymbol{u} \in \mathbb{R}^m$ is the new input Also in this case, in the presence of model uncertainties (represented as in eq. 2.13), the closed loop dynamics will be different with respect to the expected one, for both the linearized degree of freedom and the remaining zero 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 perturbation acting respectively on the active and passive subsystems.

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 is described an iterative scheme for concurrent planning and control. At its core there is a *learning process* 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 eq. 5.4–eq. 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 , 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 the robot tracks $\boldsymbol{q}^{\text{ref}}(t)$

under the action of a PFL control law (given by eq. 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 are employed Gaussian Processes regressors (Sec. 3.3.3) 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 (3.54). 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 $\boldsymbol{\delta}_a = \mathbf{0}$ and $\boldsymbol{\delta}_p = \boldsymbol{\varepsilon}_p$ in eqs. (eq. 5.4–eq. 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 in eq. 5.6 is assumed that partial feedback linearization has been correctly achieved despite the 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 $\boldsymbol{\varepsilon}_a$. Moreover, the available estimate $\boldsymbol{\varepsilon}_p$ of the perturbation on the passive subsystem has been used in eq. 5.7. Upon convergence of the overall scheme, eq. 5.6 will become exact, and $\boldsymbol{\varepsilon}_p$ in eq. 5.7 will eventually be equal to $\boldsymbol{\delta}_p$. In principle, could have also included $\boldsymbol{\varepsilon}_a$ in the right-hand side of eq. 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 proved to be computationally more efficient.

Let's 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 eq. 5.6–eq. 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

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

subject to

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

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

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

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 the specific context this idea helps to diminish the number of iterations needed by the approach to converge. 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 steers the algorithm to the learning of 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 could be obtained using the variance information of the GP 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 (eq. 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, is used the nominal PFL (eq. 5.3) to compute the generalized torque 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 the short paper [70] (from the same authors) was 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, 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 $\boldsymbol{\varepsilon}_{a,k}$ follows closely the description carried out in the previous chapter in Sec. 4.3.2. In fact, from eq. 5.4 can be expressed as

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

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. Can be noted that the actual acceleration is functionally dependent through eq. 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 [71]. However, the hyperparameters of the kernel function 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 $\boldsymbol{\delta}_p$, are compared the commanded and the actual acceleration for the passive DOFs. In fact, from eq. 5.5

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

Given the numerical approximation 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

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

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. Also in this case the hyperparameters of the kernel function 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

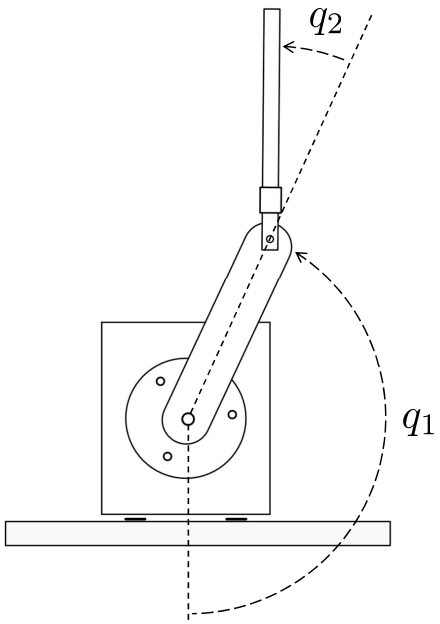

Figure 5.3. The Pendubot and its generalized coordinates.

shoulder and a passive joint at the elbow ($\boldsymbol{q}_a = q_1$ and $\boldsymbol{q}_p = q_2$). In Fig. 5.3 is shown a schematic picture of a Pendubot and the corresponding generalized coordinates.

In the following, it's addressed 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 using a Sequential Quadratic Programming (SQP) method. 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 controller

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

Convergence to the basis of attraction was obtained with 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 eq. 5.8 are chosen as $\boldsymbol{K}_P = 50$ and $\boldsymbol{K}_D = 20$, 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 $d = 180$ and chosen according to entropy maximization [46]. It should be noticed that for

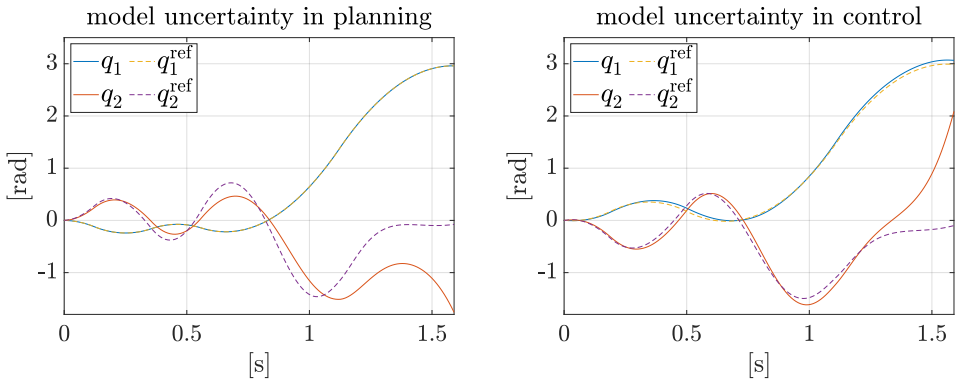

Figure 5.4. Simulation scenario 1 (*swing-up to* $\boldsymbol{q}^{\text{u-u}}$): results without learning. Left: Using the nominal model for planning and the true model for control. Right: Vice versa.

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.

The readers can refer to the following link `https://www.youtube.com/watch?v=1aKG__8gfvk&t=1s&ab_channel=RoboticsLabSapienza` for clips of all the simulations and experiments shown in the following.

5.4.1 Simulation results

Two scenarios of transfer between equilibrium states are presented. To show that the proposed method can achieve robust performance in the presence of severe model perturbations, were perturbed for control design the nominal values of the Pendubot parameters, 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 ($N = 160$).

To highlight the necessity of learning in both the planning and control phases, were preliminarily considered two complementary situations where learning is not used. Figure 5.4 (left), refers to the first situation, in which was used 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. On the contrary, 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, was 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

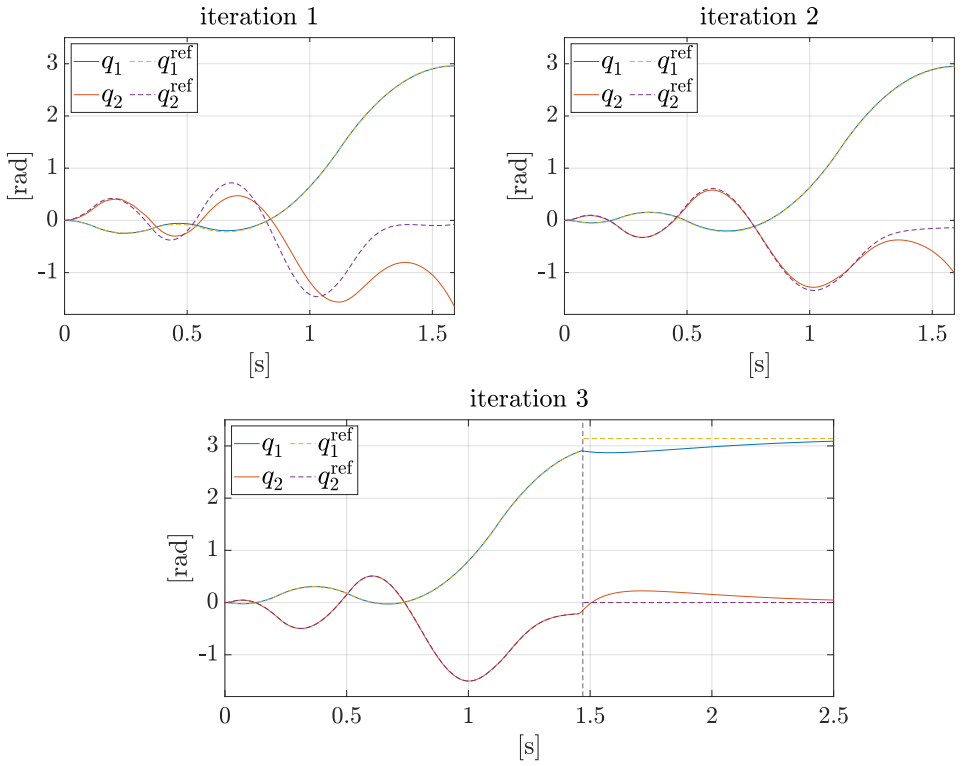

Figure 5.5. Simulation scenario 1 (*swing-up to* $\boldsymbol{q}^{\text{u-u}}$): results with the proposed approach. Just before the end of the third iteration, the state has converged to the basin of attraction of the balancing controller, which is then activated (as indicated by the vertical dashed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

of the very large model uncertainty, the learning component of the proposed 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 the result in perspective, have been applied to this scenario also the passivity-based swing-up method proposed in [59], 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 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.

Further simulations were performed 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,

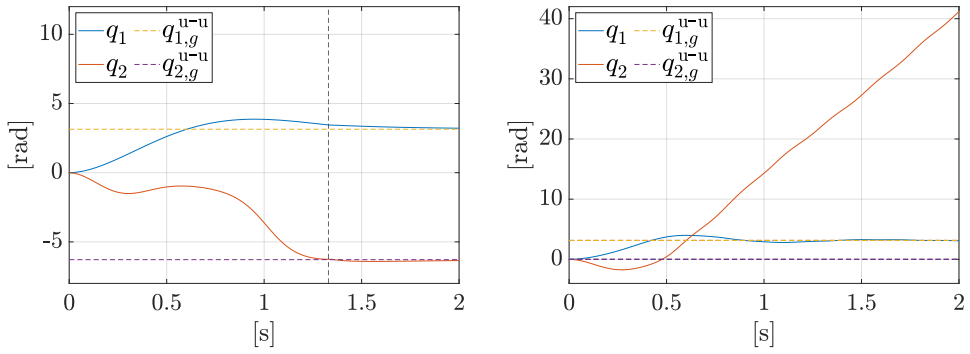

Figure 5.6. Simulation scenario 1 (*swing-up to* $\boldsymbol{q}^{\text{u-u}}$): results with the method in [59]. Left: assuming exact model knowledge the state enters the basin of attraction of the LQR controller at $t = 1.3$ s circa. Right: with the same model uncertainty of Fig. 5.5, convergence is not achieved.

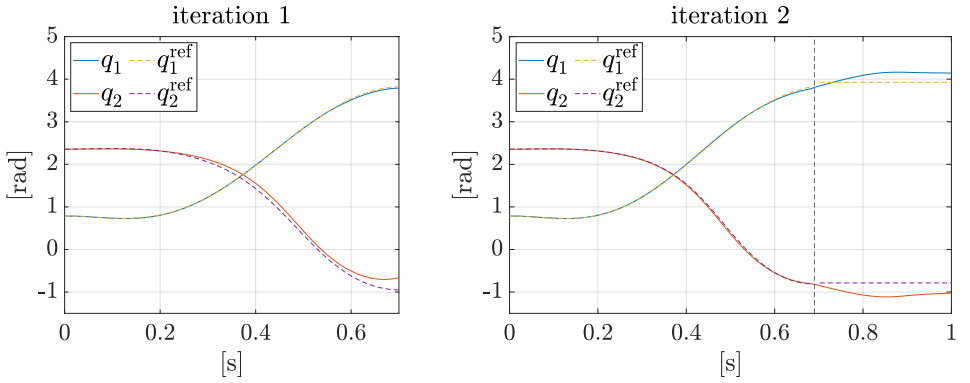

Figure 5.7. Simulation scenario 2 (*transfer between unstable equilibria*): results with the proposed approach. Two iterations are needed to achieve convergence.

a result suggesting that the method performs effectively even for higher degrees of underactuation.

5.4.2 Experimental results

The proposed method has also been tested experimentally on a Pendubot prototype, using again the nominal model for planning and control design. Joint velocities and accelerations are obtained in real-time via filtered numerical differentiation of encoder measurements. In the 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, was 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 the method to enter the basin of attraction

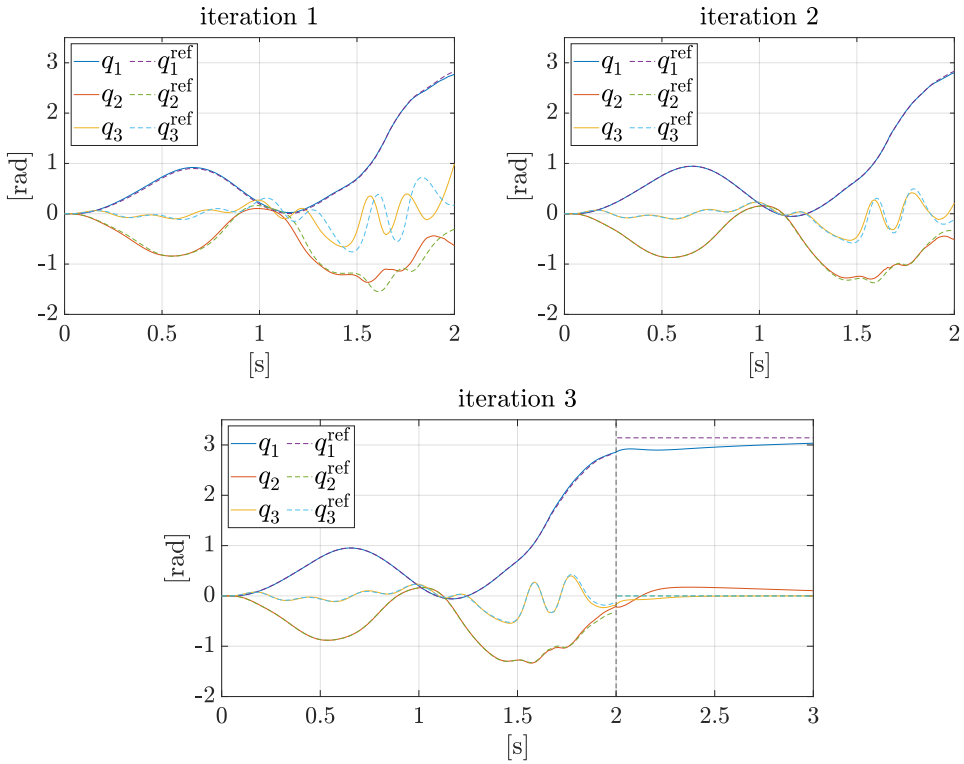

Figure 5.8. Simulation scenario 3 (*swing-up to* $\boldsymbol{q}^{\text{u-u-u}}$): results with the proposed approach. Three iterations are needed to achieve convergence.

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 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.

For comparison, Fig. 5.10 shows the experimental results obtained in this scenario with the method of [59] 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, can claimed that the learning procedure

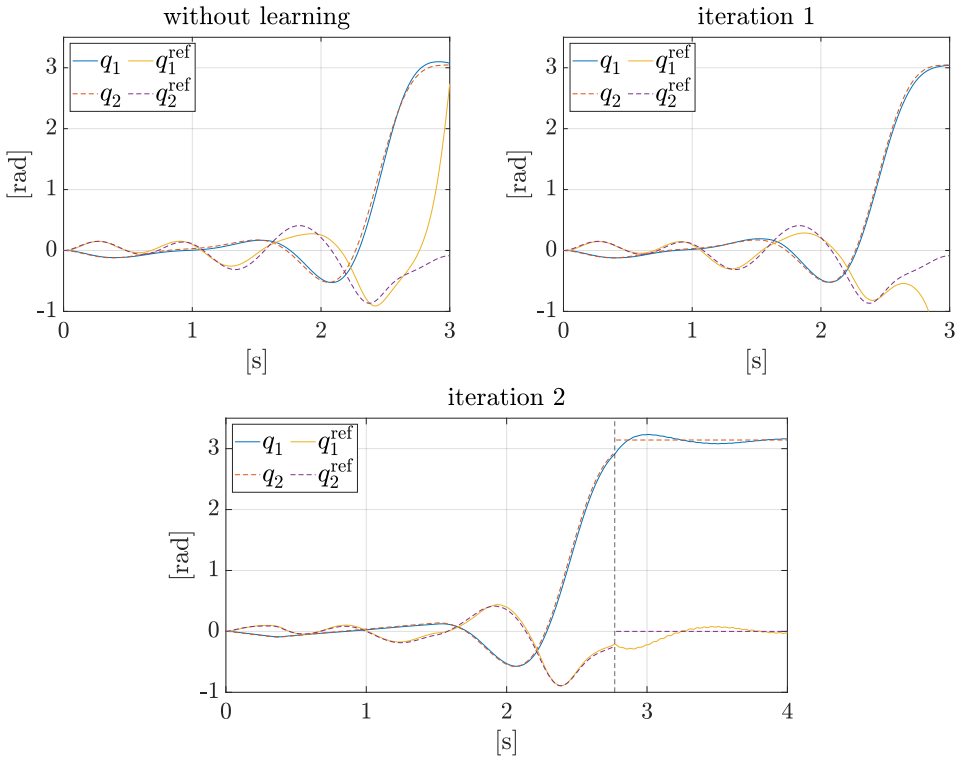

Figure 5.9. Experimental scenario 1 (*swing-up to* $\boldsymbol{q}^{\text{u-u}}$): results with the proposed approach. For comparison, the top-left image shows the results without learning, i.e., when partial feedback linearization and stable tracking for the first joint are computed on the nominal model.

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).

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	–	–

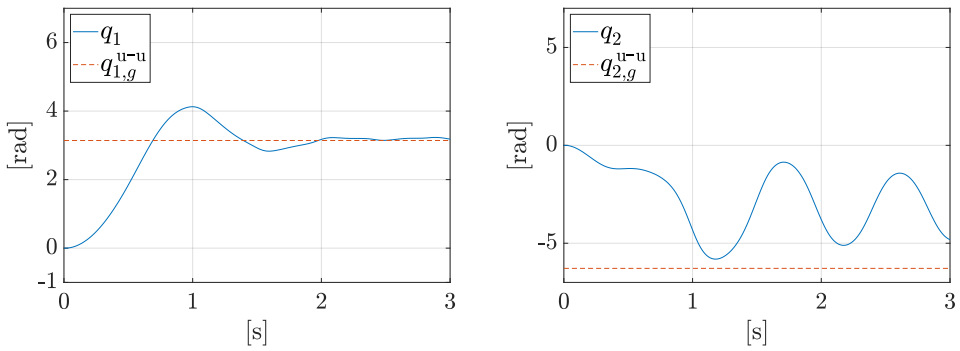

Figure 5.10. Experimental scenario 1 (*swing-up to* $\boldsymbol{q}^{\text{u-u}}$): results with the method in [59].

5.5 Chapter summary

In the chapter has been presented an iterative method for planning and controlling motions of underactuated robots in the presence of model uncertainty, built on the 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 has been 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 the iterative method, starting with considerable errors in the nominal dynamic parameters (±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 the 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 the method that deserves to be emphasized is that no torque measurement is required. In fact, only positions, 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.

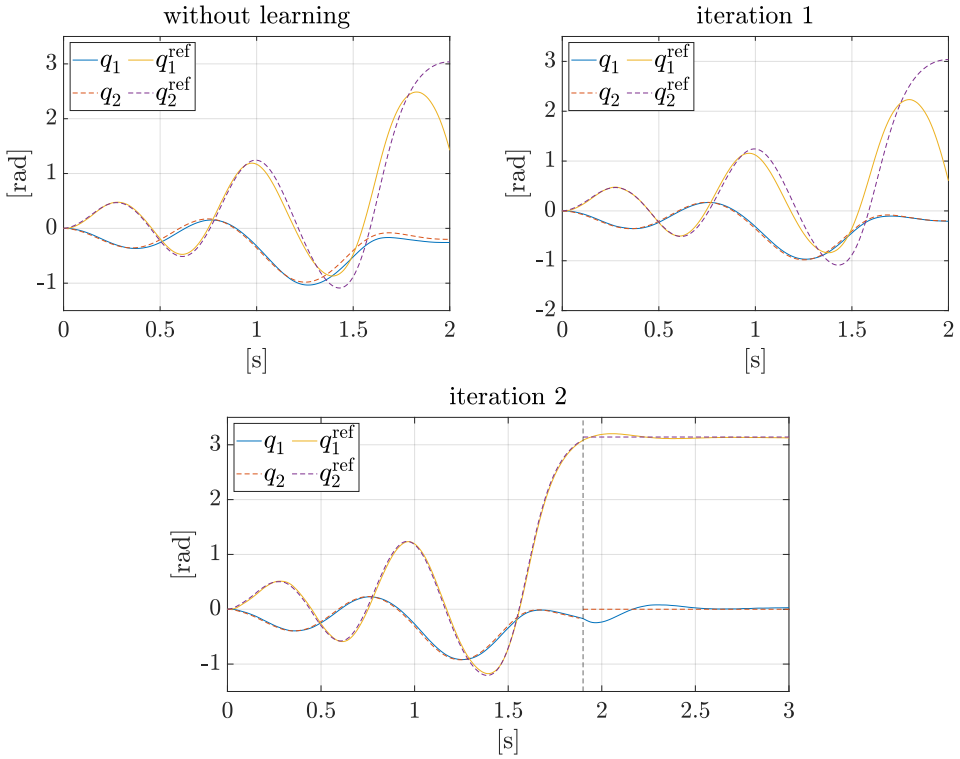

Figure 5.11. Experimental scenario 2 (*swing-up to* $\boldsymbol{q}^{\text{d-u}}$): results with the proposed approach. Just before the end of the second iteration the state converges to a region where the LQR controller can be successfully activated.

Chapter 6

Online learning and optimization for redundant robots

6.1 Motivation and contribution

As stated in previous chapters, redundancy is a condition quite dependent on the task that the robot has to realize. Robot manipulators are keen on executing particular job, usually performed through the end effector, such as grasping or moving parts. In particular, the latter can be linked to the classic trajectory tracking task; although, differently from the previous platforms (fully and underactuated), this goal is realized in the Cartesian space, also called Workspace. From this condition can be defined the redundancy of the robot: the task in fact has at most 3 degrees of freedom, while usually an antropomorph robot manipulator (such as KUKA LWR4+) have 6 or 7 actuated generalized coordinates. This means that for a generic position of the end effector there is an entire subspace of robot configurations that generates the same pose. In general, ensuring accurate trajectory tracking in Cartesian space is quite important, specifically for high precision task (such as robotic surgery); indeed, there exist control approaches that can guarantee very good tracking assuming perfect knowledge of the system model. In real application scenarios, however, this condition is hardly met, due to the presence of uncertain dynamic parameters (e.g. payload of uncertain mass) and unmodeled dynamics (e.g. friction underestimated). Since eventually inverse kinematics generates a reference acceleration to be tracked in joint space, in principle a simple idea would be to just ignore the redundancy and apply the approached proposed in 4). However, the extension of the just mentioned approach to the case of robots kinematically redundant presents some issues. In fact, if not properly managed by the control scheme, task redundancy usually prevents the generation of cyclic motions in the joint space in response to cyclic tasks [72], resulting in a continuous exploration of the input space. This behavior increases grandly the prediction errors and thus the number of steps required for convergence by a learning algorithm (referred to as Learning Transient, LT), drastically reducing control performance. A way to overcome this issue and confine the actual input space visited by the system (and so reducing the uncertainty in the prediction) is the steering of the redundant DOFs towards already explored joint space regions. A similar approach was realized in [73], with a strong difference. In that work the

goal was the reconstruction of the direct kinematics of the system, so the input optimization took in account just the robot configuration. This allowed to punctually select the new configuration that the robot has to visit to not enlarge the predictive variance of the Gaussian process. In the case of the model learning of the robot dynamics, the input consists in both position, velocities and accelerations. This aggravates the optimization largely and, more important, requires to carefully choice of the input to be optimized (this will be investigated in detail next sections). In [74] a similar approach but more sophisticated was employed in order to evaluate the dynamics of a tendon driven surgical robot, a very redundant robot. Differently from the previous case robot dynamics is considered (at least to introduce constraints on the motors torque), but still the transition model is a double integrator and the mapping between the torque exerted and the system is approximated as a function of only position and velocities of the robot, in a quasistatic approach. The null space optimization in this case is implicit, since the model predictive controller (which role is to completely control the system) exploit this redundancy to assure (at least softly) the constraints satisfaction. In the proposed approach instead some interesting properties, that in the previous case were obtained through direct optimization, such as reduced control effort and velocities, arises almost naturally. In fact the correction almost totally compensates the missing torque in the feedforward component, limiting the exploration in the input space and so reducing the motion excitement. The proposed method is able to assure precise tracking of a Cartesian trajectory for a redundant robot manipulators. This is obtained through the nominal feedback linearization of the task dynamics plus a learning based correction to recover the missing torque. Moreover, in order to reduce largely the learning transient, an optimization of the null space self motion is executed to minimize the local predictive variance of the Gaussian Process. This procedure, in general quite slow and including nonlinear solvers, is realized through an approximate method called linGP (Sec. 3.3.4), which transforms it to a classic quadratic problem.

6.2 Problem formulation

Let's consider again the actual dynamics of a robot manipulators (eq. 2.4)

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} \tag{6.1}$$

The application of an approximate feedback linearization on the robot (eq. 2.16) consists in the following closed loop

$$\ddot{\boldsymbol{q}} = \boldsymbol{u} + \boldsymbol{\delta}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \tag{6.2}$$

Including now Cartesian task defined as

$$\boldsymbol{h}(\boldsymbol{q}, \boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{0} \tag{6.3}$$

through differentiation is obtained the eq. 2.21

$$\ddot{\boldsymbol{x}} - \boldsymbol{J}(\boldsymbol{q})\ddot{\boldsymbol{q}} - \dot{\boldsymbol{J}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} = \boldsymbol{0} \tag{6.4}$$

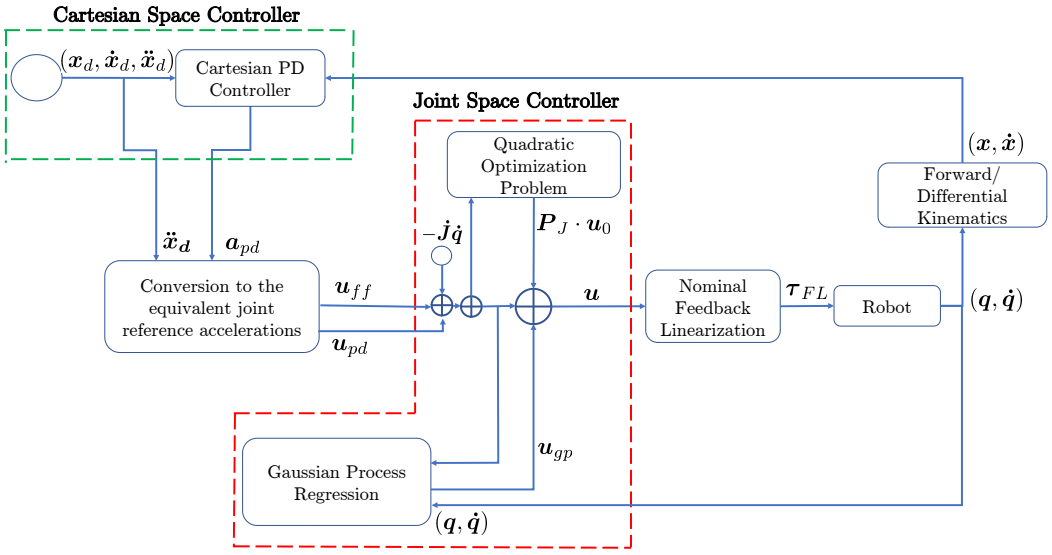

Figure 6.1. Block diagram of the proposed algorithm. The green block represents the Cartesian space controller that provides the reference Cartesian acceleration that, after the conversion to a joint space acceleration through the inverse kinematics, is provided to the red block, the Joint space controller. This block compensates the task nonlinearity, provides the GP correction, computes the optimal self motion and eventually allocates the final reference joints acceleration to the nominal feedback linearization controller.

Through the merging of the approximate feedback linearized system with the task dynamics results

$$\ddot{\boldsymbol{x}} - \boldsymbol{J}(\boldsymbol{q})\boldsymbol{u} - \boldsymbol{J}(\boldsymbol{q})\boldsymbol{\delta}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) - \dot{\boldsymbol{J}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} = \boldsymbol{0} \tag{6.5}$$

At this point the nominal input $\boldsymbol{u} = \boldsymbol{J}^{-1}\left(\dot{\boldsymbol{J}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \ddot{\boldsymbol{x}}_d + \boldsymbol{K}_P(\boldsymbol{x}_d - \boldsymbol{x}) + \boldsymbol{K}_D(\dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}})\right)$ is not anymore sufficient to linearize the Cartesian dynamics, since

$$\ddot{\boldsymbol{x}} = \ddot{\boldsymbol{x}}_d + \boldsymbol{K}_P(\boldsymbol{x}_d - \boldsymbol{x}) + \boldsymbol{K}_D(\dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}}) + \boldsymbol{\delta}_{Cartesian}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \tag{6.6}$$

so a stable second order dynamics perturbed by the nonlinear term $\boldsymbol{\delta}_{Cartesian}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) = -\boldsymbol{J}(\boldsymbol{q})\boldsymbol{\delta}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u})$.

6.3 The proposed approach

This approach is an extension of the work presented in Ch. 4, inheriting many features such as the dataset collection procedure and the idea of controlling online the system to correctly perform a feedback linearization in the joint space. Still, the generation of the reference acceleration is completely different, since focused on the closed loop task dynamics and on trying to reduce the learning transient via an optimized self motion. In this section the inherited features are not repeated, while the differences are detail investigated, such as the control architecture, the use of the linearized GP and its exploitation for the redundancy resolution.

6.3.1 Dataset collection procedure

The procedure is equivalent to the fully actuated case, investigated in detail in Sec. 4.3.2.

6.3.2 Control architecture

Similarly to what developed in Ch. 4, the scheme is tailored to learn and compensate online the missing component of the nominal torque. This is realized again fitting the Gaussian process to the collected dataset and at each control step querying the regressor to provide the correction term $\boldsymbol{\varepsilon}$. Although the framework is far more complex: can be considered in fact as a cascade of controllers that eventually provides a reference joint acceleration to the feedback linearizing controller. It will discussed from the higher to the lower level. The first block acting is the Cartesian Space Controller, which generates the nominal reference acceleration $\boldsymbol{a}$ whose intent is obtaining an asymptotically stable closed-loop error dynamics along the desired task trajectory $\boldsymbol{x}_d \in \mathbb{R}^m$. It is composed by two terms, such that $\boldsymbol{a} = \boldsymbol{a}_{ff} + \boldsymbol{a}_{pd}$ where $\boldsymbol{a}_{ff} = \ddot{\boldsymbol{x}}_d$ is the task feedforward component and $\boldsymbol{a}_{pd}$ defines the Cartesian feedback action

$$\boldsymbol{a}_{pd} = \boldsymbol{K}_P(\boldsymbol{x}_d - \boldsymbol{x}) + \boldsymbol{K}_D(\dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}}), \tag{6.7}$$

Then these Cartesian accelerations are transformed in a joint space reference (respectively $\boldsymbol{u}_{ff}$ and $\boldsymbol{u}_{pd}$) through the pseudoinversion of the Jacobian matrix (so via inverse kinematics), summed obtaining $\boldsymbol{u}_{ref}$ and provided to the Joint Space Controller. In this block many components acts simultaneously, each one presenting a contribution for the final expression:

- The joint space reference $\boldsymbol{u}_{ref}$ is redefined including a new additive term $\boldsymbol{u}_{ref} = \boldsymbol{u}_{ref} - \dot{\boldsymbol{J}}(\boldsymbol{q})\dot{\boldsymbol{q}}$ to cancel the nonlinear term in the task dynamics
- The resolution of an optimization problem (discussed in detail in next sections) returns the optimal acceleration $\boldsymbol{u}_0$
- The Gaussian process is queried using the following input $\boldsymbol{x}_* = (\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}_{query} = \boldsymbol{u}_{ref} + \boldsymbol{u}_0)$, providing the corrective term $\boldsymbol{u}_{gp} = \boldsymbol{\mu}_{\boldsymbol{x}_*}$.

Summing up all these terms is obtained the formula representing the commanded joints acceleration

$$\boldsymbol{u} = \boldsymbol{u}_{ref} + \boldsymbol{P}_J\boldsymbol{u}_0 + \boldsymbol{u}_{gp} = -\boldsymbol{J}^\dagger(\boldsymbol{q})\dot{\boldsymbol{J}}(\boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{u}_{pd} + \boldsymbol{u}_{ff} + \boldsymbol{P}_J\boldsymbol{u}_0 + \boldsymbol{u}_{gp} \tag{6.8}$$

in which the matrix $\boldsymbol{P}_J = \boldsymbol{I} - \boldsymbol{J}^\dagger\boldsymbol{J} \in \mathbb{R}^{n\times n}$ is the null-space projector of the task. The final step consists in delivering this final reference $\boldsymbol{u}$ to the nominal feedback linearization controller

$$\hat{\boldsymbol{\tau}}_{FL} = \hat{\boldsymbol{M}}(\boldsymbol{q})\boldsymbol{u} + \hat{\boldsymbol{c}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \hat{\boldsymbol{g}}(\boldsymbol{q}) \tag{6.9}$$

This scheme is accurately shown in Fig. 6.1.

6.3.3 Redundancy resolution and linGP

As mentioned in the previous sections, the redundancy of the system with respect to the Cartesian task causes a behaviour very inconvenient for the learning convergence. In fact the not uniqueness between the task value and the configuration space induces a non repetitive reference acceleration in the joint space. This can be observed easily for the first order inverse kinematic. In fact, looking at the definition of pseudoinversion given in eq. 2.31, is evident how for a given task velocity the solution depends largely from the robot configuration. In the same way, for the second order inverse kinematics, the redefinition of the term $\ddot{\boldsymbol{x}} - \dot{\boldsymbol{J}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} = \ddot{\boldsymbol{r}}$ grants the formulation of a similar problem

$$\begin{aligned} &\boldsymbol{J}(\boldsymbol{q})\ddot{\boldsymbol{q}} = \ddot{\boldsymbol{r}} \rightarrow \\ &\ddot{\boldsymbol{q}} = \boldsymbol{J}^{\#}\ddot{\boldsymbol{r}} \end{aligned} \tag{6.10}$$

obtaining again a non-unique pseudoinverse solution, this time in acceleration [75]. This is a very important difference between the method in Ch. 4, that considers only the joint space (without any kinematic inversion) and the proposed one. In line with what was said before, repetitive tasks don't induce a periodic motion in the joint space and consequently in the input space, slowing down the convergence of the learning machinery and reducing the precision and the adherence of the predictions. This is quite undesirable for an algorithm which goal is the convergence during the motion execution. These concepts can be quantitatively represented with an important additional information that a GP query provides, the predictive variance, that quantifies how much the guesses of the algorithm are trustworthy. Ideally this prediction variance should be low in already visited areas of the input space, certifying that the distribution of the target is well approximated around there, while higher in new unexplored regions. As mentioned, in the control architecture the query point of the regressor is the follow:

$$\boldsymbol{x}_* = (\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}_{query}) \tag{6.11}$$

The main idea is shape the acceleration input $\boldsymbol{u}_{query} = \boldsymbol{u}_{ref} + \boldsymbol{P}_J \boldsymbol{u}_0$ in such that the predictive variance σ_*^2 of the GP around $\boldsymbol{x}_*$ is reduced. This is obtained through a classic redundancy resolution approach: at first is solved the optimization problem, in order to find the acceleration that minimizes locally the specific cost functions and then, in order to assure the task constraints, the solution if projected to the closes solution belonging to the feasible set. These class of methods are defined locally, since joints position and velocity are considered fixed and the only decision variable consists in the instantaneous joints acceleration. This limits largely the variance reduction: what the optimization roughly realizes is assigning an acceleration similar to the previous ones (usually not too large). This technique improves the performance, but still in a limited way. Similarly to [76], a method to reduce more effectively the overall variance during the motion is the introduction at each step of a short preview. In fact through the discrete integration of the dynamics (via Euler's algorithm), next state can be approximately evaluated and inserted in the optimization procedure. In this way, an optimal solution accounting both the variance derived by the actual acceleration and the next state can influence largely the input space motion. Formally

speaking, the problem can be summarized as follows

$$\begin{aligned} \min_{\boldsymbol{u}0} \quad & \alpha\ \sigma_t^2(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{u}_{query,t}) + \beta\ \sigma_{t+1}^2(\boldsymbol{q}_{t+1}, \dot{\boldsymbol{q}}_{t+1}, \boldsymbol{u}_{query,t+1}) \\ \text{subject to} \quad & \boldsymbol{u}_{query,t} = \boldsymbol{u}_{ref} + \boldsymbol{u}_0 \\ & \ddot{\boldsymbol{q}}_t = \boldsymbol{u}_{query,t} \\ & \dot{\boldsymbol{q}}_{t+1} = \dot{\boldsymbol{q}}_t + \ddot{\boldsymbol{q}}_t Ts \\ & \boldsymbol{q}_{t+1} = \boldsymbol{q}_t + \dot{\boldsymbol{q}}_t Ts + \ddot{\boldsymbol{q}}_t Ts^2 \end{aligned} \tag{6.12}$$

in which α, β are scalar weights associated with the different components of the loss function, the state at time $t+1$ is obtained through discrete integration and Ts is the sampling time used for discretizing the system. Despite of this, the aforementioned formulation exhibits a consistency issue. In fact, differently from the positions and velocities at next time step, that are quantities obtained through time integration (continuous or discrete), acceleration is instead (at least for 2-th order systems) an instantaneous quantity. So, is not possible a priori to assign a value for $\boldsymbol{u}_{query,t+1}$ if not specifically commanding it in the subsequent time. Consequently, the actual decision variable, $\boldsymbol{u}_0$ can influence just partially the input vector at time $t+1$. On the other hand, the covariance function represents a distance in a more complex space (eq. 3.39, features space) and the reducing of the covariance between components of the input space (such as positions and velocities) instead of all vectors is also advisable. This is particularly evident for stationary kernel, depending only from the distance between the input vectors. Heuristically this can be represented as follows. Let's consider two couple of vectors in the input space $\{\boldsymbol{x}_a, \boldsymbol{x}_b\}$ and $\{\boldsymbol{x}_{a,opt}, \boldsymbol{x}_b\}$, each one in the following form $\boldsymbol{x} = (\boldsymbol{x}_{first}, \boldsymbol{x}_{last})$, of a suitable dimension N (uninfluent in this context). The peculiarity of the vector $\boldsymbol{x}_{a,opt}$ is the fact that the first component coincides with the one of $\boldsymbol{x}_b$, so $\boldsymbol{x}_{a,opt,first} = \boldsymbol{x}_{b,first}$ while its second component is the same of the not optimized version $\boldsymbol{x}_{a,opt,last} = \boldsymbol{x}_{a,last}$. The squared euclidean distance (but the argument is valid in general) between $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ is the following

$$\begin{aligned} & d_2^2(\boldsymbol{x}_a, \boldsymbol{x}_b) = \sum_{i=1}^{N} (\boldsymbol{x}_{i,a} - \boldsymbol{x}_{i,b})^2 = \sum_{first} (\boldsymbol{x}_{i,a} - \boldsymbol{x}_{i,b})^2 + \sum_{last} (\boldsymbol{x}_{i,a} - \boldsymbol{x}_{i,b})^2 = \\ & d_2^2(\boldsymbol{x}_{a,first}, \boldsymbol{x}_{b,first}) + d_2^2(\boldsymbol{x}_{a,last}, \boldsymbol{x}_{b,last}) \geq d_2^2(\boldsymbol{x}_{a,last}, \boldsymbol{x}_{b,last}) = d_2^2(\boldsymbol{x}_{a,opt}, \boldsymbol{x}_b) \end{aligned} \tag{6.13}$$

This means that also just reducing the distance between some components of the two vectors(in this case for sake of simplicity zeroing) reduces the overall distance between vectors. So, steering the input vector at the step $t+1$ in order to reduces (at least for the position and velocity components) its distance with respect to all the dataset can be important for variance minimization. Grant that, since the nonlinearity with respect of the input of the predictive variance, the next's step variance cannot be explicitly evaluated. Indeed in this circumstance the use of linGP is crucial. As explained in Sec. 3.3.4, this technique approximates the original Gaussian process by another one, whose predictive mean and variance becomes respectively linear and quadratic with respect to a new input, redefined as the distance w.r.t. to the approximation point. This transforms the variance at time t and time $t+1$ in a quadratic function, simplifying the problem's resolution and the partition of the different contributions. In this way, the effects of the last component of the input

$\boldsymbol{u}_{query,t+1}$ can just be ignored and the preview assumes consistency and only the effects generated by the actual decision variable $\boldsymbol{u}_0$ are considered. In next sections the mathematical formulation of these approximations is presented.

Prediction variance at time t

Considering the predictive variance at time t and given the linearization point $\boldsymbol{x}_{lin} = (\boldsymbol{q}_{lin}, \dot{\boldsymbol{q}}_{lin}, \ddot{\boldsymbol{q}}_{lin})^T$, a new input $\hat{\boldsymbol{x}}$ can be expressed as follows

$$\hat{\boldsymbol{x}}_t(\ddot{\boldsymbol{q}}_t) = \begin{pmatrix} 1 \\ \boldsymbol{\Delta}_{x,t} \end{pmatrix} \tag{6.14}$$

where

$$\Delta_{x,t} = \begin{pmatrix} \boldsymbol{q}_t - \boldsymbol{q}_{lin} \\ \boldsymbol{q}_t - \dot{\boldsymbol{q}}_{lin} \\ \ddot{\boldsymbol{q}}_t - \ddot{\boldsymbol{q}}_{lin} \end{pmatrix} \tag{6.15}$$

Being $\boldsymbol{q}_t$ and $\dot{\boldsymbol{q}}_t$ fixed, the eq. (6.14) can be split into a constant (given the approximation point) and a time-varying part, i.e.

$$\hat{\boldsymbol{x}}_t(\ddot{\boldsymbol{q}}_t) = \boldsymbol{b}_t + \boldsymbol{w}_t = \begin{pmatrix} 1 \\ \boldsymbol{q}_t - \boldsymbol{q}_{lin} \\ \dot{\boldsymbol{q}}_t - \dot{\boldsymbol{q}}_{lin} \\ -\ddot{\boldsymbol{q}}_{lin} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \ddot{\boldsymbol{q}}_t \end{pmatrix} \tag{6.16}$$

with $\ddot{\boldsymbol{q}}_t$ being the decision variable (with respect to eq. 6.12 corresponds to $\boldsymbol{u}_{query,t}$). Following, it does hold

$$\begin{aligned} \sigma_t^2 &= (\boldsymbol{b}_t + \boldsymbol{w}_t)^T \boldsymbol{V} (\boldsymbol{b}_t + \boldsymbol{w}_t) \\ &= \boldsymbol{b}_t^T \boldsymbol{V} \boldsymbol{b}_t + \boldsymbol{w}_t^T \boldsymbol{V} \boldsymbol{b}_t + \boldsymbol{b}_t^T \boldsymbol{V} \boldsymbol{w}_t + \boldsymbol{w}_t^T \boldsymbol{V} \boldsymbol{w}_t \end{aligned} \tag{6.17}$$

and, taking into account the symmetry of $\boldsymbol{V}$

$$\sigma_t^2 = \boldsymbol{b}_t^T \boldsymbol{V} \boldsymbol{b}_t + 2\boldsymbol{b}_t^T \boldsymbol{V} \boldsymbol{w}_t + \boldsymbol{w}_t^T \boldsymbol{V} \boldsymbol{w}_t \tag{6.18}$$

Hence, the variance expressed in eq. 6.18 is a complete quadratic problem with respect to $\boldsymbol{w}_t$. As a recall, the goal was to find $\ddot{\boldsymbol{q}}_t$ such that the variance gets minimized. For this purpose, the constant term, i.e. $\boldsymbol{b}_t^T \boldsymbol{V} \boldsymbol{b}_t$, can be dropped out, and the term $\boldsymbol{w}_t$ can be expressed as

$$\boldsymbol{w}_t = \boldsymbol{A}_t \ddot{\boldsymbol{q}}_t \tag{6.19}$$

where, by assuming $\boldsymbol{q} \in \mathbb{R}^n$, it results

$$\boldsymbol{A}_t = \begin{pmatrix} \boldsymbol{0}_{1\times n} \\ \boldsymbol{0}_{n\times n} \\ \boldsymbol{0}_{n\times n} \\ \boldsymbol{I}_{n\times n} \end{pmatrix} \tag{6.20}$$

As a result the predictive variance at time t can be represented as a quadratic loss with respect to $\ddot{\boldsymbol{q}}_t$

$$\sigma_t^2 = \frac{1}{2} \ddot{\boldsymbol{q}}_t^T \boldsymbol{H}_t \ddot{\boldsymbol{q}}_t + \boldsymbol{f}_t^T \ddot{\boldsymbol{q}}_t \tag{6.21}$$

where

$$\boldsymbol{H}_t = 2\boldsymbol{A}_t^T \boldsymbol{V} \boldsymbol{A}_t \qquad \boldsymbol{f}_t = 2\boldsymbol{b}_t^T \boldsymbol{V} \boldsymbol{A}_t \tag{6.22}$$

Prediction variance at time t+1

The formulation is similar to before: the new input $\hat{\boldsymbol{x}}$ at time $t+1$ can be written as

$$\hat{\boldsymbol{x}}_{t+1}(\ddot{\boldsymbol{q}}_t) = \begin{pmatrix} 1 \\ \boldsymbol{\Delta}_{x,t+1} \end{pmatrix} \tag{6.23}$$

where

$$\boldsymbol{\Delta}_{x,t+1} = \begin{pmatrix} \boldsymbol{q}_{t+1} - \boldsymbol{q}_{lin} \\ \dot{\boldsymbol{q}}_{t+1} - \dot{\boldsymbol{q}}_{lin} \\ \ddot{\boldsymbol{q}}_{t+1} - \ddot{\boldsymbol{q}}_{lin} \end{pmatrix} \tag{6.24}$$

and $\boldsymbol{q}_{t+1}$, $\dot{\boldsymbol{q}}_{t+1}$ and $\ddot{\boldsymbol{q}}_{t+1}$ represent respectively the joint configuration, velocity ad acceleration at the next step $t+1$. As mentioned before in the nonlinear formulation, the quantities $\boldsymbol{q}_{t+1}$ and $\dot{\boldsymbol{q}}_{t+1}$ can be obtained through numerical integration, as follows

$$\begin{aligned} \dot{\boldsymbol{q}}_{t+1} &= \dot{\boldsymbol{q}}_t + \ddot{\boldsymbol{q}}_t Ts \\ \boldsymbol{q}_{t+1} &= \boldsymbol{q}_t + \dot{\boldsymbol{q}}_t Ts + \ddot{\boldsymbol{q}}_t Ts^2 \end{aligned} \tag{6.25}$$

Including the numerical integration in eq. 6.23, it results as

$$\hat{\boldsymbol{x}}_{t+1}(\ddot{\boldsymbol{q}}_t) = \begin{pmatrix} 1 \\ \boldsymbol{q}_t + \dot{\boldsymbol{q}}_t Ts + \ddot{\boldsymbol{q}}_t Ts^2 - \boldsymbol{q}_{lin} \\ \dot{\boldsymbol{q}}_t + \ddot{\boldsymbol{q}}_t Ts - \dot{\boldsymbol{q}}_{lin} \\ \ddot{\boldsymbol{q}}_{t+1} - \ddot{\boldsymbol{q}}_{lin} \end{pmatrix} \tag{6.26}$$

Again, being the current joint position and velocity fixed, eq. 6.14 can be split into a constant and a time-varying part, i.e.

$$\hat{\boldsymbol{x}}_{t+1}(\ddot{q}_t) = \boldsymbol{b}_{t+1} + \boldsymbol{w}_{t+1} = \begin{pmatrix} 1 \\ \boldsymbol{q}_t + \dot{\boldsymbol{q}}_t Ts - \boldsymbol{q}_{lin} \\ \dot{\boldsymbol{q}}_t - \dot{\boldsymbol{q}}_{lin} \\ -\ddot{\boldsymbol{q}}_{lin} \end{pmatrix} + \begin{pmatrix} 0 \\ \ddot{\boldsymbol{q}}_t Ts^2 \\ \ddot{\boldsymbol{q}}_t Ts \\ \ddot{\boldsymbol{q}}_{t+1} \end{pmatrix} \tag{6.27}$$

The variance matrix, since symmetric quadratic matrix mixing different components, can be decomposed in the following way:

$$\boldsymbol{V} = \begin{pmatrix} \boldsymbol{V}_{q,q} & \boldsymbol{V}_{q,\dot{q}} & \boldsymbol{V}_{q,\ddot{q}} \\ \boldsymbol{V}_{\dot{q},\dot{q}} & \boldsymbol{V}_{\dot{q},\dot{q}} & \boldsymbol{V}_{\dot{q},\ddot{q}} \\ \boldsymbol{V}_{\ddot{q},q} & \boldsymbol{V}_{\ddot{q},\dot{q}} & \boldsymbol{V}_{\ddot{q},\ddot{q}} \end{pmatrix} \tag{6.28}$$

Since acceleration at time $t+1$ is not directly modifiable through the actual optimization input (in this case $\ddot{\boldsymbol{q}}_t$), here is realized the important approximation mentioned above (that in the nonlinear case couldn't be realized). It consists in removing the submatrices which represent the interactions of the next step istantaneous accelerations $\ddot{\boldsymbol{q}}_{t+1}$ with the other components of the input vector (positions and velocities) and substituting them with matrices of zeros (of suitable dimension). After that, the new variance matrix becomes as follows

$$\hat{\boldsymbol{V}} = \begin{pmatrix} \boldsymbol{V}_{q,q,} & \boldsymbol{V}_{q,\dot{q}} & \boldsymbol{0} \\ \boldsymbol{V}_{\dot{q},\dot{q}} & \boldsymbol{V}_{\dot{q},\dot{q}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix} \tag{6.29}$$

In such way the effects of the acceleration assigned at the time step $t+1$ are removed. Is very important the fact that, since the original matrix $\boldsymbol{V}$ is symmetric, real and positive definite, all principals minors are positive definite. Knowing that $\det(\hat{\boldsymbol{V}}) = \boldsymbol{M}_{3,3}(\boldsymbol{V}) > 0$ (since principal minor) and $\hat{\boldsymbol{V}}$ is still symmetric, so also the new matrix is positive definite, so satisfying all the properties required to a variance matrix. Clearly the resulting variance will be in general different; nevertheless, according to its interpretation as a distance in the feature space, considering the effects of just a part of the state at time $t+1$ (only positions and velocities) can improve the performance (as actually results also in numerical simulations). Hence the approximated next step predictive variance can be written as

$$\begin{aligned}\sigma^2_{t+1} &= (\boldsymbol{b}_{t+1} + \boldsymbol{w}_{t+1})^T \hat{\boldsymbol{V}} (\boldsymbol{b}_{t+1} + \boldsymbol{w}_{t+1}) \\ &= \boldsymbol{b}^T_{t+1}\hat{\boldsymbol{V}}\boldsymbol{b}_{t+1} + \boldsymbol{w}^T_{t+1}\hat{\boldsymbol{V}}\boldsymbol{b}_{t+1} + \boldsymbol{b}^T_{t+1}\hat{\boldsymbol{V}}\boldsymbol{w}_{t+1} + \boldsymbol{w}^T_{t+1}\hat{\boldsymbol{V}}\boldsymbol{w}_{t+1} \\ &= \boldsymbol{b}^T_{t+1}\hat{\boldsymbol{V}}\boldsymbol{b}_{t+1} + 2\boldsymbol{b}^T_{t+1}\hat{\boldsymbol{V}}\boldsymbol{w}_{t+1} + \boldsymbol{w}^T_{t+1}\hat{\boldsymbol{V}}\boldsymbol{w}_{t+1}\end{aligned} \tag{6.30}$$

where the symmetry of $\hat{\boldsymbol{V}}$ has been used. As in the previous case, the term $\boldsymbol{b}^T_{t+1}\hat{\boldsymbol{V}}\boldsymbol{b}_{t+1}$ is constant and does not affect the minimization problem, then can be dropped out. Moreover, since last row and column of $\boldsymbol{V}$ have been removed, both the last component of $\boldsymbol{w}_{t+1}$ and $\boldsymbol{b}_{t+1}$ are irrelevant. To recover previous notation, is arbitrarily considered the last component of $\boldsymbol{w}_{t+1}$ ($\ddot{\boldsymbol{q}}_{t+1}$) as zero, in such a way to obtain again an expression of the form

$$\boldsymbol{w}_{t+1} = \boldsymbol{A}_{t+1}\ddot{\boldsymbol{q}}_t \tag{6.31}$$

where

$$\boldsymbol{A}_{t+1} = \begin{pmatrix} \boldsymbol{0}_{1\times n} \\ \boldsymbol{I}_{n\times n}Ts^2 \\ \boldsymbol{I}_{n\times n}Ts \\ \boldsymbol{0}_{n\times n} \end{pmatrix} \tag{6.32}$$

As a result the predictive variance at time $t+1$ can be represented as a quadratic loss with respect to $\ddot{\boldsymbol{q}}_t$

$$\sigma^2_{t+1} = \frac{1}{2}\ddot{\boldsymbol{q}}^T_t \boldsymbol{H}_{t+1}\ddot{\boldsymbol{q}}_t + \boldsymbol{f}^T_{t+1}\ddot{\boldsymbol{q}}_t \tag{6.33}$$

where

$$\boldsymbol{H}_{t+1} = 2\boldsymbol{A}^T_{t+1}\hat{\boldsymbol{V}}\boldsymbol{A}_{t+1} \qquad \boldsymbol{f}_{t+1} = 2\boldsymbol{b}^T_{t+1}\hat{\boldsymbol{V}}\boldsymbol{A}_{t+1} \tag{6.34}$$

Complete problem and null space damping

In the previous sections both the predictive variances at time t and $t+1$ have been transformed in quadratic functions. In such a way, the problem in eq. 6.12 is expressed now as

$$\begin{aligned} \min_{\boldsymbol{u}0} \quad & \alpha\frac{1}{2}\ddot{\boldsymbol{q}}^T_t(\alpha\boldsymbol{H}_t + \beta\boldsymbol{H}_{t+1})\ddot{\boldsymbol{q}}_t + (\alpha\boldsymbol{f}^T_t + \beta\boldsymbol{f}^T_{t+1})\ddot{\boldsymbol{q}}_t \\ \text{subject to} \quad & \boldsymbol{u}_{query,t} = \boldsymbol{u}_{ref} + \boldsymbol{u}_0 \\ & \ddot{\boldsymbol{q}}_t = \boldsymbol{u}_{query,t} \end{aligned} \tag{6.35}$$

in which α and β represents the weights acting on the different components of the loss function. The resolution of this problem generates an optimal acceleration $\boldsymbol{u}_0^{opt}$. As known from classical redundancy resolution with quadratic cost, the final result can be analytically expressed as the pseudoinverse of the task reference acceleration summed to the solution of the quadratic problem projected in the Jacobian null space so, with the same notation of eq. 6.8

$$\boldsymbol{u}_0 = \boldsymbol{u}_{ref} + \boldsymbol{P}(\boldsymbol{q})\boldsymbol{u}_{opt} \tag{6.36}$$

with $\boldsymbol{P}(\boldsymbol{q})$ projector in the null space. Eventually, as usual in second order redundancy resolution scheme, is added a damping term in the null space, to introduce self motion stability and limit oscillating behaviours

$$\boldsymbol{u}_0^{opt} = \boldsymbol{u}_0^{opt} - \boldsymbol{K}_v\dot{\boldsymbol{q}} \tag{6.37}$$

6.4 Simulation results

In this section are reported the results of simulations performed applying the proposed method on a simulated KUKA LWR iiwa 4+ manipulator, a robot with n=7 revolute joints, performing a Cartesian trajectory tracking task. The tracking capabilities of the simulated robot are analyzed, comparing the performance obtained with different type of controller:

- nominal Cartesian FL
- nominal Cartesian FL + GP correction
- nominal Cartesian FL + GP correction + self motion optimization
- nominal Cartesian FL + GP correction + self motion optimization + damping in the null space

In order to establish a discrepancy between the nominal robot model (reported in in [54, 55]) and the actual one, is introduced a mismatch between the masses and the inertia of around 30% (randomly distributed, both increasing and shrinking) and the friction term (composed by both a linear and nonlinear component) is excluded from the nominal model. Indeed, the kinematic of the robot is granted as known (classic assumption for robot manipulators). The end effector initial state has been assumed coincident with the reference one, to avoid large initial tracking error. The system is numerically integrated via Euler's integration algorithm, assuming a sample and control period coincident of 10 milliseconds. The addressed problem consists in the tracking a Cartesian horizontal circular trajectory at constant height, of radius R = 15 cm in T = 2 s. The gains used in the Cartesian controller (eq. 6.7) are $\boldsymbol{K}_P = \text{diag}\{100, 100, 100\}$, $\boldsymbol{K}_D = \text{diag}\{20, 20, 20\}$, while the gain on the self motion damping is $\boldsymbol{K}_V = \text{diag}\{10, 10, 10, 10, 10, 10, 10\}$

The first tested controller corresponds to the nominal Cartesian FL, so the expression showed in eq. 6.8 without the learning adjustment and the self motion optimization

$$\boldsymbol{u} = \boldsymbol{u}_{ref} = -\boldsymbol{J}^{\dagger}(\boldsymbol{q})\dot{\boldsymbol{J}}(\boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{u}_{pd} + \boldsymbol{u}_{ff} \tag{6.38}$$

Due to the presence of model uncertainties, the tracking performance is drastically reduced, as shown in Fig. 6.2(red line).On the other hand, the control effort and the joint velocities of the system are increased, since the controller tries to recover the cumulative tracking error by increasing the magnitude of the commanded torque (Fig. 6.3). The second controller consists in the first one with the adding of the GP correction, in order to compensate the not precise FL

$$\boldsymbol{u} = -\boldsymbol{J}^{\dagger}(\boldsymbol{q})\dot{\boldsymbol{J}}(\boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{u}_{pd} + \boldsymbol{u}_{ff} + \boldsymbol{u}_{gp} \tag{6.39}$$

Due to the presence of the learning transient in such a fast motion, the tracking performance w.r.t. the Cartesian FL controller is improved mainly toward the end of the trajectory execution (Fig. 6.2, blue line). Similarly, both the the control effort and joints velocities are reduced when the learning algorithm starts the convergence phase (Fig. 6.3). The third controller includes the null-space contribution

$$\boldsymbol{u} = -\boldsymbol{J}^{\dagger}(\boldsymbol{q})\dot{\boldsymbol{J}}(\boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{u}_{pd} + \boldsymbol{u}_{ff} + \boldsymbol{u}_{gp} + \boldsymbol{P}(\boldsymbol{q})\boldsymbol{u}_0 \tag{6.40}$$

The performance in this case is highly improved, in both Cartesian error and motion smoothness, reducing drastically the joint velocities and the control effort. Is important to notice that the predictive variance (and consequently the standard deviation, which is the square root) thanks to the optimization is drastically reduced, as visible in Fig. 6.5. This is an index on the fact that motion is limited in the input space. Eventually the last controller includes the optimal acceleration redefinition, according to eq. 6.37, so introducing a null space damping term. This extra term reduces more the overall velocities and consequently also the velocity error; because of this, a lower feedback component is generated by the Cartesian PD, facilitating the GP correction. Also in this case the standard deviation is very low (Fig. 6.5), but in the same order of the one obtained with just the self motion optimization.

6.5 Chapter summary

In this chapter has been proposed an online learning-based controller to realize accurate Cartesian trajectory tracking for redundant robots in presence of model uncertainties, guaranteeing high performance also during the learning transient through a self- motion optimization. The approach was tested on a simulated KUKA LWR4+ robot through a comparative study, that showed how the proposed control framework achieves the best behavior in terms of tracking error reduction, motion smoothness, and reduced control effort.

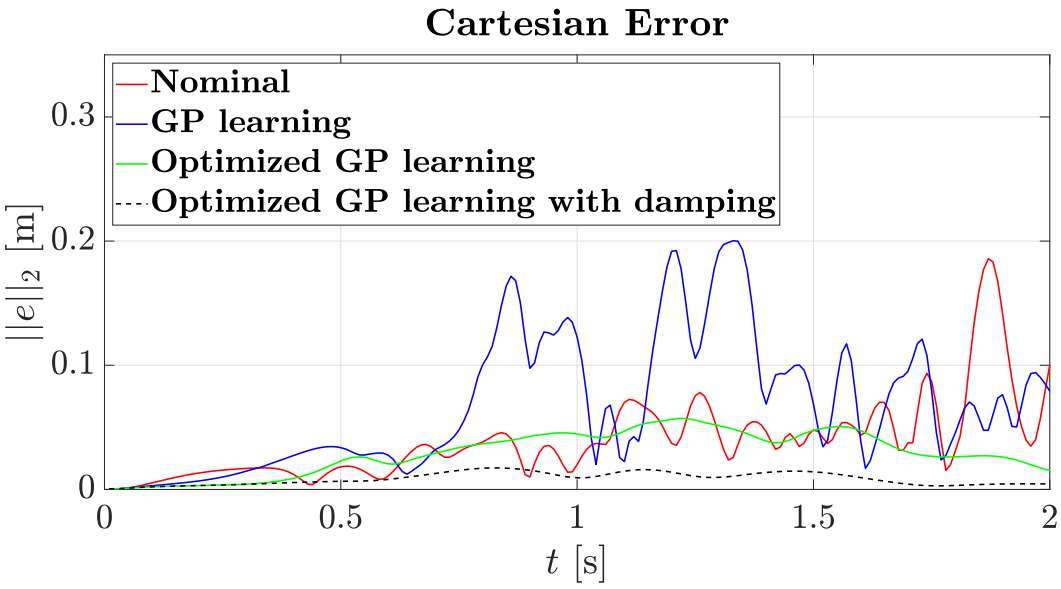

Figure 6.2. Plot representing the punctual norm of Cartesian error vector vs time. Is evident how, differently from the approach presented in Ch. 4, the tracking of a Cartesian trajectory generates a continuous exploration in the joint space, reducing the performance of the learning correction (blue line); paradoxically the nominal controller (red) reaches a better performance. However, With the introduction of an optimized self motion (green line), the error is drastically reduced. Eventually the use of a damping (black dashed line) improves slightly the performance, obtaining an overall error reduction of order 10 with respect to the nominal controller.

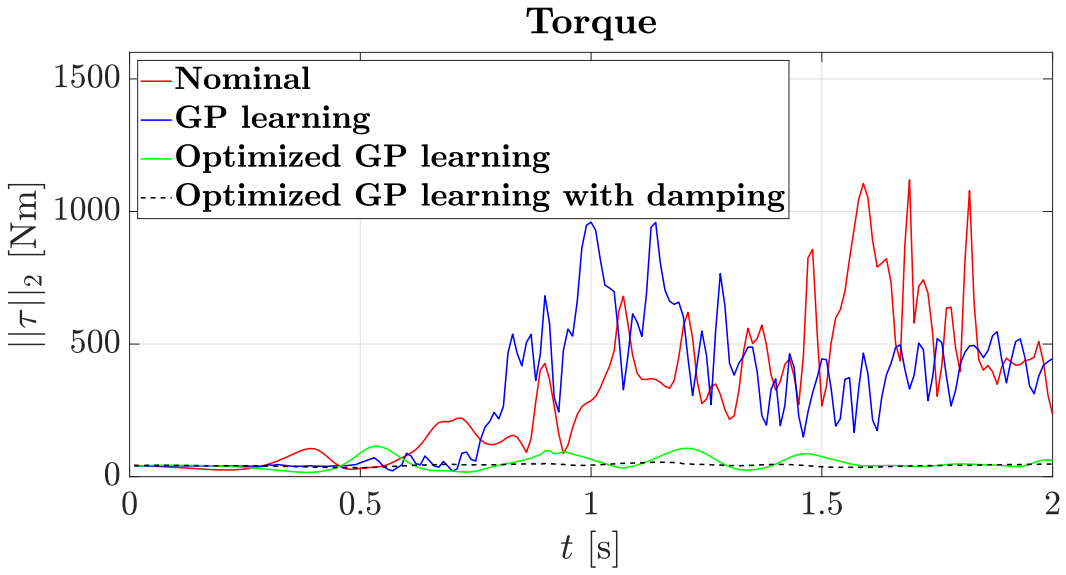

Figure 6.3. Plot representing the punctual norm of the torque vector vs time. Since the long learning transient, such as Fig. 6.2, the controller produces a larger input, for recovering via closed loop feedback the erroneous FL torque. This is quite evident in both the nominal (red line) and the one corrected via a GP without self motion (blue line). Again, self motion optimization with and without damping (respectively black dashed and green line) reduces drastically the control effort.

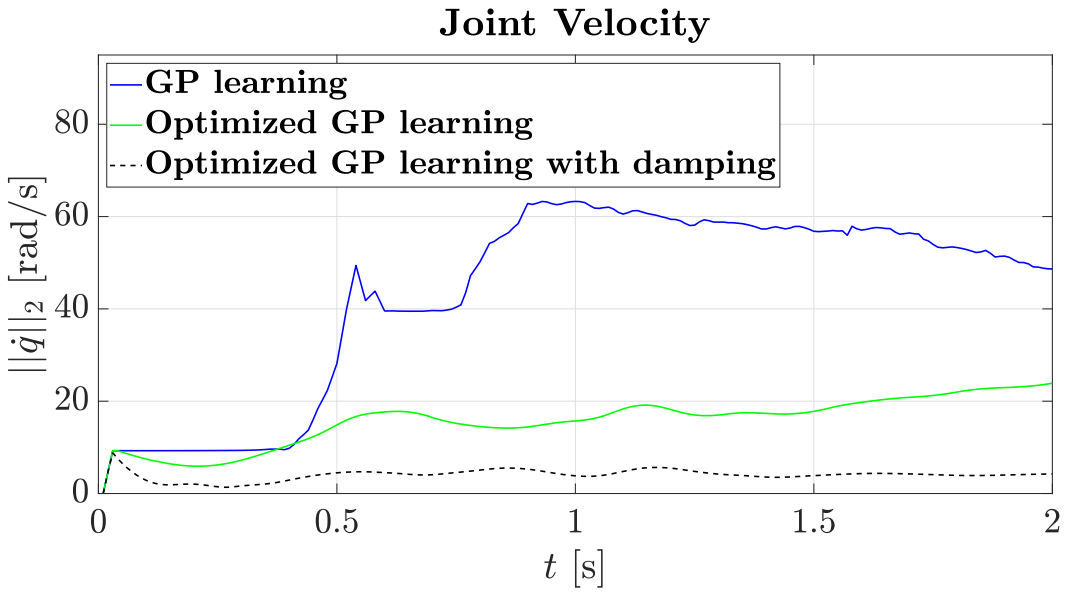

Figure 6.4. Plot representing the punctual norm of the velocity vector vs time. As previous plots is evident the improvement of both optimized and optimized with damping controller. The norm of the velocity vector of the nominal controller is not introduced, since completely out of scale with respect to the others.

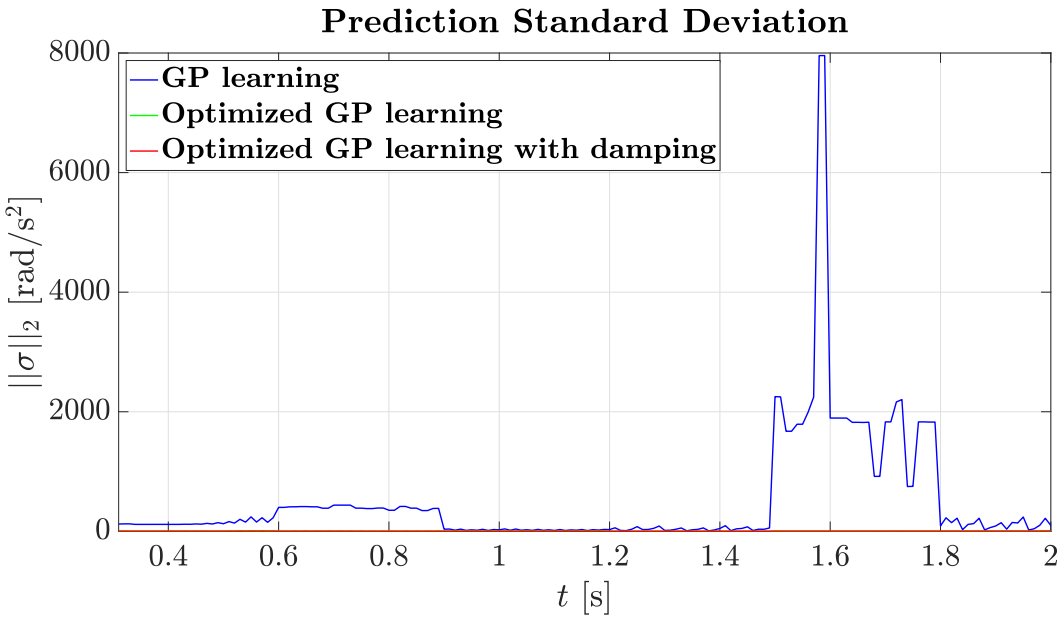

Figure 6.5. Plot representing the punctual norm of the predictive standard deviation vs time. This quantity can be evaluated only with controllers GP based, so nominal Cartesian FL is not considered. The value of variance obtained during the motion with just the GP correction (blue line) is quite higher with respect to the optimized one (green line) and the optimized plus damping (red line). Green and red lines almost overlap and given the scale it is impossible to distinguish them.

Chapter 7

Conclusions

In this thesis new methodologies has been presented that, according to the different robot structure, allowed the execution of a precise trajectory tracking through model learning techniques. In the introductory chapter was discussed why the model based controllers have improved largely the performance of robot controllers; after that has been presented a brief overview around how the problem of a correct model reconstruction was tackled classically in robotics and through new model learning techniques. The next chapter was instead focused on providing preliminary knowledge around two class of model based controllers: Computed Torque and Model Predictive Controller. Computed torque techniques are based on the idea that through the model knowledge is possible to compute a torque able to shape the closed loop dynamics (via a nonlinear feedback) and transforming it to a desired one. In the case the resulting dynamics is linear, the technique is also called Feedback Linearization (FL). Then this approach has been investigated in detail, for both fully and under actuated systems, characterizing their difference and the controller's range of application. After that has been presented redundant robots, defining in what consists the redundancy and analyzing the application of FL to modify the task dynamics. In the final part of the chapter has been briefly introduced the Model Predictive Controller, a very different control approach with respect to Computed Torque, based on the idea that the exploitation of the knowledge of the robot's dynamics allows the generation of a preview of the system's evolution, and according to it, the finding of a local optimal control input.

The succeeding chapter have introduced preliminary notions around model learning in robotics. In the beginning were analyzed classical model learning techniques, consisting in the parametrization of the robot dynamics through the regressor matrix and the dynamic coefficients. After that was deeply investigated the method used to retrieve these set of parameters, called identification, showing in the first place the classic approaches and then analyzing a more specific case in which the author provided a contribution; to conclude was then exposed a very known control technique based on this scheme, called adaptive Control. The second part of the chapter was instead focused on machine learning based approaches to the resolution of the model learning problem. All of these techniques are primarily based on the idea of represent the model retrieval as a regression problem. Simpler techniques, such as Least Square (recovering also previous robotics identification methods), Max-

imum Likelihood Estimation and Maximum a Posteriori Estimation were discussed. Eventually, in the final part of the chapter was presented a large section around Gaussian process regression. This technique is very important, since consists in the baseline method on top of which all the novelty of the manuscript were based. At the beginning of the section has been illustrated the classical formulation, analyzing the Bayesian approach and linking it with the Stochastic Processes theory. After that, another section was dedicated on the analysis of covariance functions, showing their important features. Then advantages and weaknesses of the method have been illustrated, in order to introduce approximate Gaussian process regression as a good trade off between computational tractability and precision. To conclude the chapter, were presented two approximation techniques: sparse approaches, that relies on a subset of specific quantities to reduce the dimension of kernel or dataset and linearized gaussian process, a linearization technique very convenient since preserving important stochastic properties.

After these very general chapters, has been shown the contributions of the author to the resolution of the trajectory tracking problem in robotics with model learning techniques.

In Ch. 4 has been exposed a trajectory tracking controller composed by a nominal feedback linearization with a Gaussian procession, in which the latter provides an extra torque able to online reconstruct the correction required to correctly linearize the system. The architecture of the controller is composed of an higher level Model Predictive Controller, which exploits a double integrator transition model (assuming that the feedback linearization is correctly performed) and by a lower level controller block (FL controller plus the Gaussian process correction), which task is the exact linearization of the system. Peculiarity of the method is the fact that the correction term is reconstructed using the controllability Gramian, instead of torque measurements (as common in literature). To demonstrate the effectiveness of the method has been then tested on a simulated 7-DOFs Kuka LBR robot.

In Ch. 5 the previous method was extended in the case of underactuated robots. Since the presence of underactuation, the precedent schema couldn't be directly used to manage all the DOFs of the system. A classic architecture for controlling underactuated robots has been exploited: an offline planning of a feasible trajectory followed by the motion execution, in which the trajectory is tracked through a model based controller (Partial Feedback Linearization). Preliminarly was shown that the realization of the assigned task depends not only from the correct online trajectory tracking, but also from the availability of a feasible trajectory for the system. That implied that the acquired knowledge had to been inserted also in the planning phase, not just in the online control phase. In order to do this the scheme of offline planning/online tracking was iterated, updating the transition model of the planner with the new information and correcting online the tracking controller (PFL). With just few iterations the method was able to solve different tasks, also including an high level of uncertainty (around 30% of the robot dynamics). The algorithm has been tested on simulated 2R robot called Pendubot, composed by an actuated joint followed by a passive one, realizing two different tasks. Furthermore, to show the autonomy of the method with respect to the platform, the method was proved on a simulated 3R robot with the last two joints passive, obtaining similar results. Eventually the method has been tried on a real robot, so a realistic scenario,

showing high performance and converging after only two iterations.

Finally, in Ch. 6, the approach has been extended to redundant robots. In this context the goal was the tracking of given a Cartesian trajectory: this has been basically realized using a FL in task space with the inclusion of a GP correction on the nominal torque (so extending the method presented in Ch. 4. Despite of this, the main difference with respect to the previous case was the fact that the reference joints acceleration were generated trough an inverse kinematics procedure. This dramatically slowed down the convergence of the learning algorithm: in fact, the inverse kinematics in general doesn't not provide cyclicality of the motion, dragging the regressor in a continuous exploration phase. In order to limit this exploration and so reduce the learning transient, the redundancy has been solved through a self motion, which goal was the minimization of the prediction variance. To both speeding up the inference (since all of this is realized online) and at the same time adding consistency in the optimization, the linGP was employed. This approximation of the Gaussian process was very convenient and coherent with all the literature of redundancy resolution: in such a way the prediction variance became a quadratic form, which optimization is a very classic redundancy resolution problem (Ch. 6). The method has been tested on a simulated KUKA LWR4+ robot, showing its performance: high accuracy in the Cartesian tracking control and contemporary reduction of control effort and joint velocities.

7.1 Future outlook

All these methods relies on the idea of correcting online (when possible) the nominal feedback linearization, providing a suitable modification of the reference acceleration. Nevertheless, according to the specific application, different future improvements may be realized. Regarding the basics implementation shown in Ch. 4, the online correction for fully actuated systems, possible extensions of the proposed method are the testing of the technique on different real robots manipulator and the use of the GP predictive variance as proxy of the learning's convergence, to employ the MPC constraints in robust [77] or stochastic way [22]. Another possible extension, not explicitly accounted but quite easy to employ, is the use of this method for system that cannot be statically but only dynamically feedback linearized (such as drones or elastic joints manipulators). Regarding instead the method proposed in Ch. 5, a future direction is considering the problem of guaranteeing hard constraints during the entire learning transient; this can be realized by explicitly taking into account the covariance of the uncertainty during the planning phase. Again, as for the fully actuated approach, the test of the method on different platforms (in this case underactuated), namely quadrotor UAVs and humanoids, is a scheduled step.

Considering eventually the approach exposed in Ch. 6, differently from the aforementioned ones, it partially uses the predictive variance in order to realize an optimal self motion. In such a way the algorithm exploits the available information to locally obtain a better performance. A possible development of the method may be the introduction of an active learning strategy, in order to generalize on different tasks while reducing even further the learning transient. Again as previous techniques, experiments on different redundant platforms is an expected step.

Bibliography

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

[2] C. Gaz, E. Magrini, and A. De Luca, "A model-based residual approach for human-robot collaboration during manual polishing operations," *Mechatronics*, vol. 55, pp. 234–247, 2018.

[3] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving Rubik's cube with a robot hand," 2019. [Online]. Available: http://arxiv.org/abs/1910.07113

[4] 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.

[5] 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.

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

[7] H. Kimura, T. Yamashita, and S. Kobayashi, "Reinforcement learning of walking behavior for a four-legged robot," in *Proceedings 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, C. D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, 2019-01-16.

[10] J. Peters, D. D. Lee, J. Kober, D. Nguyen-Tuong, J. A. Bagnell, and S. Schaal, "Robot learning," in *Springer Handbook of Robotics*, 2nd ed., B. Siciliano and O. Khatib, Eds. Springer, 2016, pp. 357–394.

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

[12] 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 Conference on Robot Learning*, vol. 100, 2020, pp. 1359–1368.

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

[14] M. Ficorilli, M. Capotondi, V. Modugno, and A. D. Luca, "Exploiting robot redundancy for online learning and control," in *Proceedings Italian Conference on Robotics and Intelligent Machines*, 2022, pp. to–be–defined.

[15] D. E. O. Roy Featherstone, in *Handbook of Robotics*, 2nd ed., B. Siciliano and O. Khatib, Eds. Springer, 2016, pp. 38–66.

[16] 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.

[17] R. Tedrake, *Underactuated Robotics*, 2022. [Online]. Available: https://underactuated.csail.mit.edu

[18] A. A. M. Stefano Chiaverini, Giuseppe Oriolo, "Redundant robots," in *Springer Handbook of Robotics*, 2nd ed., B. Siciliano and O. Khatib, Eds. Springer, 2016, pp. 242–221.

[19] D. E.Kirk, *Optimal Control Theory: An Introduction.* Dover Publications, 1998.

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

[21] D. Q. Mayne, M. M. Seron, and S. V. Raković, "Robust model predictive control of constrained linear systems with bounded disturbances," *Automatica*, vol. 41, no. 2, p. 219–224, feb 2005.

[22] M. C. Campi, S. Garatti, and M. Prandini, *Scenario Optimization for MPC.* Cham: Springer International Publishing, 2019, pp. 445–463.

[23] C. Della Santina, M. G. Catalano, and A. Bicchi, *Soft Robots.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2020, pp. 1–15.

[24] C. Gaz, M. Cognetti, A. Oliva, P. Robuffo Giordano, and A. De Luca, "Dynamic identification of the franka emika panda robot with retrieval of feasible parameters using penalty-based optimization," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4147–4154, 2019.

[25] J.-J. Slotine and L. Weiping, "Adaptive manipulator control: A case study," *IEEE Transactions on Automatic Control*, vol. 33, no. 11, pp. 995–1003, 1988.

[26] M. Gautier and W. Khalil, "Exciting trajectories for the identification of base inertial parameters of robots," in *Proceedings IEEE Conference on Decision and Control*, 1991, pp. 494–499 vol.1.

[27] M. Pennese, C. Gaz, M. Capotondi, V. Modugno, and A. D. Luca, "Identification of robot dynamics from motor currents/torques with unknown signs," in *Proceedings Italian Conference on Robotics and Intelligent Machines*, 2021, pp. 125–129.

[28] N. Hawes, J. L. Wyatt, M. Sridharan, M. Kopicki, S. Hongeng, I. Calvert, A. Sloman, G.-J. Kruijff, H. Jacobsson, M. Brenner, D. Skočaj, A. Vrečko, N. Majer, and M. Zillich, "The playmate system," in *Cognitive Systems*, H. I. Christensen, G.-J. M. Kruijff, and J. L. Wyatt, Eds. Springer Berlin Heidelberg, 2010, pp. 367–393.

[29] 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.

[30] 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.

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

[32] J. Umlauft, T. Beckers, M. Kimmel, and S. Hirche, "Feedback linearization using Gaussian processes," in *Proceedings IEEE Conference on Decision and Control*, 2017, pp. 5249–5255.

[33] A. Yesildirek and F. Lewis, "Feedback linearization using neural networks," in *Proceedings IEEE International Conference on Neural Networks*, vol. 4, 1994, pp. 2539–2544 vol.4.

[34] M. I. Jordan and D. E. Rumelhart, "Forward models: Supervised learning with a distal teacher," *Cognitive Science*, vol. 16, no. 3, pp. 307–354, 1992.

[35] S. L. Brunton, M. Budišić, E. Kaiser, and J. N. Kutz, "Modern Koopman theory for dynamical systems," *SIAM Review*, vol. 64, no. 2, pp. 229–340, 2022.

[36] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz, "On dynamic mode decomposition: theory and applications," *Journal of Computational Dynamics*, vol. 1, no. 2, pp. 391–421, 2014.

[37] D. Bruder, X. Fu, R. B. Gillespie, C. D. Remy, and R. Vasudevan, "Data-driven control of soft robots using Koopman operator theory," *IEEE Transactions on Robotics*, vol. 37, no. 3, pp. 948–961, 2021.

[38] A. Bjorck, *Numerical methods for least squares problems.* Philadelphia, PA: SIAM, 1996.

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

[40] C. T. H. Baker, *The Numerical Treatment of Integral Equations.* Oxford University Press, 1978.

[41] 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.

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

[43] 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.

[44] A. Gahlawat, P. Zhao, A. Patterson, N. Hovakimyan, and E. A. Theodorou, "L1-gp: L1 adaptive control with Bayesian learning," in *Proceedings of Conference on Learning for Dynamics and Control*, vol. 120, 2020, pp. 826–837.

[45] 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.

[46] N. Lawrence, M. Seeger, and R. Herbrich, "Fast sparse Gaussian process methods: The informative vector machine," in *Advances in Neural Information Processing Systems*, S. Becker, S. Thrun, and K. Obermayer, Eds., vol. 15. MIT Press, 2002.

[47] M. Deisenroth and S. Mohamed, "Expectation propagation in Gaussian process dynamical systems," in *Proceedings Advances in Neural Information Processing Systems*, vol. 25, 2012.

[48] T. X. Nghiem, "Linearized Gaussian processes for fast data-driven model predictive control," in *Proceedings IEEE American Control Conference*, 2019, pp. 1629–1634.

[49] E. Solak, R. Murray-Smith, W. E. Leithead, D. J. Leith, and C. E. Rasmussen, "Derivative observations in Gaussian process models of dynamic systems," in *Proceedings International Conference on Neural Information Processing Systems*, 2002, p. 1057–1064.

[50] 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.

[51] E. Magrini and A. De Luca, "Hybrid force/velocity control for physical human-robot collaboration tasks," in *Proceedings IEEE International Conference on Intelligent Robots and Systems*, 2016, pp. 857–863.

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

[53] 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.

[54] 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.

[55] 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.

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

[57] M.Spong, "Underactuated mechanical systems," in *Control Problems in Robotics and Automation*, 1998, pp. 135–150.

[58] I. Fantoni and R. Lozano, "The Pendubot system," in *Nonlinear Control for Underactuated Mechanical Systems*, 2002, pp. 53–72.

[59] 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.

[60] 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 IEEE Conference on Decision and Control*, 2006, pp. 6138–6143.

[61] 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.

[62] 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.

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

[64] 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 International Conference on Intelligent Robots and Systems*, 2017, pp. 51–58.

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

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

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

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

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

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

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

[72] A. De Luca, L. Lanari, and G. Oriolo, "Control of redundant robots on cyclic trajectories," in *Proceedings IEEE International Conference on Robotics and Automation*, 1992, pp. 500–506.

[73] Y. Yun and A. D. Deshpande, "Control in the reliable region of a statistical model with Gaussian process regression," in *Proceeding IEEE International Conference on Intelligent Robots and Systems*, 2014, pp. 654–660.

[74] 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.

[75] A. De Luca and G. Oriolo, "Issues in acceleration resolution of robot redundancy," *IFAC Proceedings Volumes*, vol. 24, no. 9, pp. 93–98, 1991.

[76] K. Al Khudir, G. Halvorsen, L. Lanari, and A. De Luca, "Stable torque optimization for redundant robots using a short preview," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2046–2053, 2019.

[77] 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.