SAPIENZA
UNIVERSITÀ DI ROMA

On-Line Learning Control of Redundant Robots with Uncertain Dynamics

Faculty of Information Engineering, Informatics, and Statistics
Master Degree in Control Engineering

Marco Ficorilli
ID number 1797705

Advisor	Co-Advisor
Prof. Alessandro De Luca	Dr. Marco Capotondi

Academic Year 2021/2022

On-Line Learning Control of Redundant Robots with Uncertain Dynamics
Master Thesis. Sapienza University of Rome

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

Author's email: marcoficorilli98@gmail.com

“Predicting the future isn’t magic,
it’s artificial intelligence.”
(Dave Waters)

Ringraziamenti

Prima di procedere con la trattazione, vorrei dedicare qualche riga a tutti coloro che mi sono stati vicini in questo percorso di crescita personale e professionale.

Un sentito grazie al mio relatore, il professor Alessandro De Luca, per essere stato una guida fin dall'inizio di questo mio percorso. Non dimenticherò le sue lezioni, e porterò con me i suoi consigli. Nella mia crescita lei ha giocato un ruolo fondamentale, mi ha trasmesso qualcosa che non si insegna, la passione e la curiosità verso questo magnifico settore.

Vorrei ringraziare Marco Capotondi, supervisore e compagno in questi mesi dedicati alla tesi, per il supporto e i consigli, personali e professionali, che mi ha fornito. Grazie per tutto il tempo che mi hai dedicato.

Ringrazio infinitamente la mia famiglia, per aver sempre e fermamente creduto in me e nelle mie scelte. Mi avete accompagnato fin dall'inizio, in ogni passo, e la certezza che questo non cambi mai non ha prezzo. Ogni traguardo o successo deve reggersi su qualcosa, e voi siete la base di tutto.

Un ringraziamento particolare va alla mia fidanzata Giulia, che mi ha sempre messo al primo posto, in ogni occasione. Grazie per avermi insegnato a condividere ogni momento, che sia un esame, una laurea, o un piccolo traguardo di una giornata, e per far sì che ogni volta questo momento acquisisca un valore immenso.

Ringrazio i miei amici, fondamentali per tutti i momenti di svago fuori dall'università. Il rapporto maturato in questi anni è qualcosa di raro, e, nonostante gli impegni e la distanza, non mancano mai quelle uscite o quei messaggi che strappano una risata, contribuendo ad un equilibrio che viene quasi scambiato per regolarità, ma che in realtà rappresenta una delle fortune più grandi.

Infine, vorrei ringraziare i colleghi dei corsi di Control Engineering e di Artificial Intelligence and Robotics, con i quali ho condiviso non solo l'intero percorso di studi, ma anche passioni e momenti di spensieratezza. È vostro un contributo significativo alla bellezza di questi due anni.

Grazie infinite a tutti voi.

Abstract

This thesis proposes a method that merges a classical nonlinear feedback law with a machine learning technique in the presence of uncertain dynamics, with the aim of controlling a robot which is redundant with respect to a desired task. Very often a perfect knowledge of the robot dynamic model is not available, because of uncertainty in the dynamic parameters (link masses, center of masses and inertia) and due to the presence of friction which is typically hard to model. Other sources of uncertainty include the addition of an unknown payload or partial faults on the torque delivered by the actuators. To estimate on-line, i.e., during robot motion, the torque that is needed for perfect task execution, which is missing due to uncertain dynamics, a Gaussian process-based regressor has been implemented. To overcome the computational time issues associated with standard Gaussian processes, a sparse linearized version of the algorithm has been used, providing a significant improvement of the overall method. When the robot is redundant for the task, the extra degrees of freedom can be exploited to enhance performance in trajectory tracking tasks. On the other hand, by the use of the linearized Gaussian process, the prediction variance, which can be seen as a measure of the prediction error, can be rewritten as a quadratic form. Therefore, redundancy is used to locally minimize the prediction variance by formulating a Linear Quadratic optimization problem. This problem can be efficiently solved on-line, allowing for a significant speed-up with respect to its nonlinear version. The whole approach has been developed without the need of joint torque sensors, which are known to be expensive and noisy. Several simulations results are reported on different trajectory tasks performed with the 7-DOF KUKA LWR4+ robot, comparing the nominal model-based approach with the one that includes the proposed learning correction. The benefits provided by this control method are further highlighed in the presence of a more complex but unmodeled friction behavior of the robot.

Contents

Chapter 1

Introduction

This thesis deals with the implementation of an on-line learning procedure for controlling a redundant robot, with respect to some desired tasks, with uncertain dynamics.
Indeed, an uncertainty will be introduced on the mass of each link of the robot, and the resulting nominal dynamic model will be used for controlling the manipulator.
First, a model-based approach by using the Computed Torque control law will be tried, representing also the starting baseline; then, this control law will be improved by some learning-based techniques in order to achieve better performance, showing the effectiveness of the proposed approach.

1.1 Motivations

First of all, one may wonder what are the benefits that a learning-based approach guarantees if combined with other model-based approaches like the very common Feedback Linearization (FL) for controlling a robot.
In an uncertainty-free context, Feedback Linearization ideally guarantees the perfect cancellation of all nonlinearities in the robot dynamical model, and then a precise execution of the desired motion tasks. However, in order to have such performance, a very accurate dynamic model is strictly required, and often this is not available. Indeed, due to residual parametric uncertainties and unmodeled dynamics, e.g. frictions, which are often very hard to model, a full cancellation of the nonlinear terms by feedback is hardly achieved, implying a severe worsening of the performance scored in executing the desired task.
On the other hand, the model-based approach can be improved with some learning-based techniques in order to recover the error introduced by the model mismatches, then including some extra torque able to compensate for it and make the robot to execute correctly the desired motion. Of course, it is needed for this corrective torque to recover for the errors on-line, i.e. while the robot is performing its own task.

Consequently, the goal of this thesis is to develop a framework aimed at improving on-line the torque correction necessary for obtaining a perfect cancellation of all the nonlinear terms with a FL controller by learning the unmodeled dynamics, using only joint position measurements, i.e. without joint torque sensors, which are known to be expensive and noisy.

1.2 Basic Idea

The basic idea relies into considering a mismatch between the real and the nominal robot's dynamic models. This mismatch can be introduced due to identification errors, unmodeled dynamics or even the introduction of a payload at the end-effector level. Following, the nominal control law is applied to the real system together with a torque correction term needed to balance the errors thus introduced.
Hence, the key point is to find this correction torque. The main idea here is to use a data-driven approach, in particular a supervised learning technique in order to predict at best the amount of torque that is needed to compensate for the dynamic errors.

In the literature, there are several machine learning based algorithms that suits this problem, like Neural Networks, SVM or any other regression model. However, some of them require an off-line training phase for well performing, and it would be preferable to recover for this error on-line, during the robot motion, without a specific training phase before executing the desired task. For this purpose, a regressor based on Gaussian Processes (GPs) has been selected: indeed, the Gaussian Processes are well-known for their ability of scoring good performance with small datasets. Moreover, the use of a Gaussian Process returns also the prediction variance, highlighting the possibility of improving the performance by properly exploiting this information. However, more details about Gaussian Processes and the variance optimization will be delivered in the following chapters.

1.3 Thesis Structure

The reminder of the thesis is organized as follows.

The second chapter gives a detailed overview of the main problem addressed in this thesis, providing also the theoretical background about robot kinematics, dynamics and control required to fully understand the proposed topics. Moreover, the case of a dynamic model with uncertainties on the link masses is discussed and the resulting consequences are exploited.

Following, the third chapter deals with the proposed approach for solving the critical issues discussed in Chapter 2. In particular, it provides the theoretical and mathematical notions needed to characterize both the Nonlinear and the Linearized Gaussian Process regression, together with a detailed description of how to generate a suitable dataset for this problem.

Next, the fourth chapter discuss one of the main topics of this thesis, i.e. the optimization of the variance in the robot Jacobian null-space. Indeed, being the robot redundant with respect to the desired task, the remaining degrees of freedom can be employed to generate a self motion aimed at minimizing the prediction variance, and then at improving the overall performance. In particular, this kind of problem is also formulated as an efficient Quadratic problem, providing a considerable speed-up if compared with its nonlinear version.

Then, the fifth chapter presents the results of the simulation performed by applying the whole approach to the considered robot, i.e. a KUKA LWR 4+ manipulator. Here, in particular, the operational settings used for running the simulation are provided and the tasks the robot has to perform are designed. Moreover the results for each task are shown through several plots highlighting the benefits and the clear improvement obtainable by integrating the classical model-based approach with some learning-based techniques.

In the sixth chapter, instead, a more realistic case where the presence of unmodeled frictions is taken into account is reported, making the model-based approach improved with the learning correction to shine with respect to the pure model-based one. Also, some comments about the possibility of correctly estimating the unmodeled frictions are given.

Finally, the seventh chapter proposes a brief recall of the main topics, methods, analysis and considerations addressed in this thesis, together with possible future works on this topic.

Chapter 2

Problem Formulation

In this chapter, the main problem addressed in this thesis is accurately formulated. First of all, the theoretical background and the basic notations about robot kinematics, dynamics and control needed to fully understand the nature of the proposed problem are provided, exploiting also the possible uses of redundancy and the amazing performance of the model-based control laws given a perfect knowledge of the dynamic model in an uncertainty-free context. Finally, the limitations introduced by an uncertain dynamics possibly due to unmodeled dynamics and/or uncertainties in the estimated dynamic parameters, e.g. the links masses, are explicitly pointed out.

2.1 Robot Kinematics

The kinematics of robot manipulators is the study of geometric and timing aspects of robot motion, without reference to the causes producing it. Here the robot is seen as an open kinematic chain, i.e. without self loops in the kinematic structure, of rigid bodies interconnected by joints, that can be either revolute or prismatic. The study of the robot kinematics is very useful for the definition of the robot workspace and for mapping two different spaces, the joint space and the task space.
The joint space is characterized by the choice of the parameterization q, which must define the robot configuration in an unambiguous and minimal way. The number of robot joints, either rotational or translational, also set the number n of degrees of freedom (DoFs) of the robot.
The task space, instead, is characterized by the choice of the parameterization r, describing compactly the position and the orientation variables related to the required task. Depending on the nature of the desired task, the dimension m of the task itself is defined, characterizing also the minimum number of DoFs needed to perform that particular task. For example, if the desired task requires both position and orientation in the Cartesian space, then m will be equal to 6; if, instead, only the position matters, m will be set to 3. Usually, $m \leq n$, and if this relation is strict, then the robot is said to be redundant with respect to that particular task. However, the topic of redundancy will be deeply exploited later on.

About the mapping between the two spaces, if this is from the Joint to the Task space, it takes the name of Direct Kinematics, vice versa it is called Inverse Kinematics.

The Direct Kinematics problem consists into finding the end-effector pose by the available joint configuration. From a mathematical point of view, it corresponds to finding f such that $r = f(q)$. A solution for this problem always exists, it is unique and it can be computed geometrically, i.e. by inspection for structurally simple robots, or systematically, assigning frames attached to robot links and using homogeneous transformation matrices.
The Inverse Kinematics problem, instead, consists into finding the joint configuration q that would realize a given desired end-effector pose, and it can be expressed as $q = f^{-1}(r)$. Typically, this is an highly nonlinear problem, where the existence of a solution depends on the robot workspace, since the desired pose may be out of this, and there can be one or multiple solutions. About the multiplicity of solutions, some general cases can be identified:

- if $m > n$, it does not exist any solution;
- if $m = n$, either no solution exists, or there are a finite number of solutions;
- if $m < n$, either no solution exists, or, due to the kinematic redundancy for the task, an infinite set of solutions exists, and in particular ∞^{n-m} solutions.

It should be noticed that, in case of kinematic singularity, different situations may happen.
However, if at least a solution exists, it can be found either analytically, i.e. in closed form, or numerically, i.e. in iterative form. The former, if it can be found, is preferred, while the latter is certainly needed for redundant cases or when approaching singularities and it can be computed through either Gradient or Newton method.
Further details about direct and inverse kinematics can be found on the references [libro], [rob1], mentioned in the bibliography.

Finally, there is the so-called Differential Kinematics, providing the relations between motion in joint space, i.e. velocity, and motion in task space, i.e. linear or angular velocity. Basically, the instantaneous velocity mapping can be obtained either by differentiating with respect to time the direct kinematics or directly at the differential level, in a geometric way.
Following, the concept of analytical and geometric Jacobian matrices arises. In particular, the analytical Jacobian is obtained by time differentiation of the direct kinematics, i.e.

$$r = \begin{pmatrix} p \\ \phi \end{pmatrix} = f(q) \qquad \longrightarrow \qquad \dot{r} = \begin{pmatrix} \dot{p} \\ \dot{\phi} \end{pmatrix} = \frac{\partial f(q)}{\partial q}\dot{q} = J(q)\dot{q} \tag{2.1}$$

while the geometric or basic Jacobian can be computed without the use of derivatives as follows

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} J_L(q) \\ J_A(q) \end{pmatrix} \dot{q} = J(q)\dot{q} \tag{2.2}$$

where v and ω represent the linear and the angular velocity respectively. It is worth to notice that, in both cases, the Jacobian depends on the current configuration of the robot.

It is worth to look for higher-order differential kinematics. Indeed, differential relations between motion in the joint and in the task space can be characterized also at the second order, third order, and so on. In particular, let's focus to the acceleration level, where the following relation does hold:

$$\ddot{r} = J(q)\ddot{q} + \dot{J}(q)\dot{q} \tag{2.3}$$

Looking at Equation 2.3, one may notice also that the Jacobian is acting as a weight for the highest-order derivative.
However, looking at the rank of the Jacobian matrix, some interesting considerations can be done.
Given a matrix J with m rows and n columns, the Jacobian rank $\rho(J)$ is defined as the maximum number of linearly independent rows or columns. By definition, the $\rho(J)$ can be at most the minimum between m and n. It is worth to notice that, if $\rho(J) = \min(m, n)$, then J has full rank. Moreover, if J has full rank and it is squared, i.e. $m = n$, it is also invertible.
Moreover, is always possible to perform a decomposition of the robot Jacobian into linear subspaces, i.e. the range space $R(J)$ and the null-space $N(J)$, respectively called also image and kernel of J, such that the following property holds

$$R(J) + N(J^T) = \mathbf{R}^m \qquad \wedge \qquad R(J^T) + N(J) = \mathbf{R}^n \tag{2.4}$$

Then, the range space $R(J)$ is defined as the subspace of all linear combinations of the columns of J, i.e.

$$R(J) = \{v \in \mathbf{R}^m \;:\; \exists\, \xi \in \mathbf{R}^n,\; v = J\xi\} \tag{2.5}$$

and it has dimension $\dim(R(J)) = \rho(J)$. This set represents also the subspace of all generalized velocities that can be instantaneously realized by the robot end-effector when varying the joint velocities $\dot{q}$ at the current q. The null-space $N(J)$, instead, is defined as the subspace of all vectors which are zeroed by the robot Jacobian, i.e.

$$N(J) = \{\xi \in \mathbf{R}^n \;:\; J\xi = 0 \in \mathbf{R}^m\} \tag{2.6}$$

and it has dimension $\dim(N(J)) = n - \rho(J)$. If it is different from the naive solution, i.e. zero vector, then there are non-zero joint velocities $\dot{q}$ producing zero end-effector velocity, i.e. self motions. It should be noticed that, if the robot is redundant with respect to some task, the Jacobian null-space is always different from the naive set.

2.2 Kinematic Redundancy

The last kinematic recall that is needed to exploit is the kinematic redundancy.
A robot is said kinematically redundant with respect to a given task $r = f(q)$ if $n > m$. In other words, the robot has more degrees of freedom than strictly needed for executing the task. It is important to notice that redundancy always depends on the given task, and this is why one can speak about redundancy with respect to a specific task, but not about redundancy of the robot itself.
In general, there exist different types or redundancy: indeed, other than kinematic redundancy, one may have redundancy of components, like sensors or actuator, or

redundancy in the control and supervision architecture. However, in this thesis, the word redundancy is always referred to the kinematic one.

There are many possible uses of the kinematic redundancy. Just to mention, it can be used for avoiding collision with possible moving or fixed obstacles, at least locally if these obstacles are not known; similarly, it can be used for avoiding kinematic singularities: indeed, even if the total number of singularities of a redundant robot may be larger w.r.t. a non-redundant one, the number of possible ways to avoid them hugely increases. Redundancy can also be used to stay within the admissible joint ranges, by considering the joint limits as hard constraints, i.e. something that must be never violated. Other possible uses are to increase manipulability in specific directions, to uniformly distribute joint velocities and/or accelerations, or minimizing the needed motion torques, the energy consumption or the time needed for executing a specific task.
It is clear that redundancy can be used for multiple purposes, but all the objectives that one want to pursue must be quantitatively measurable, in order to write and solve a suitable optimization problem. In the case proposed in this thesis, redundancy will be exploited to minimize the prediction variances given by the Gaussian Processes.
Up to now, redundancy has shown a lot of advantages, but it has also some disadvantages. In particular, it implies a greater structural complexity of construction, meaning more actuators, more sensors, links, transmissions and, last but not least, and increase of the costs. Moreover, also the software part about inverse kinematics and motion control gets complicated.

There are different ways to handle redundancy. A first differentiation is between local and global methods. The first aims at optimizing the objective function on-line, and is usually related to solving LQ problems, while the second is an highly nonlinear method and requires to be solved off-line in a numerical way. However, let's focus on the local ones.
Essentially there can be identified three classes of local methods for solving $\dot{r} = J(q)\dot{q}$, i.e.

- Jacobian-based methods, where among infinite solutions, the one minimizing a suitable, and usually weighted, norm is chosen;
- Null-space methods, where a term is added to the Jacobian-based solution so as to not affect the execution of the task trajectory or, in other terms, a term is added in the null-space of the robot Jacobian. This implies an extra motion that allows to pursue some objective and probably formulate a linear quadratic optimization problem, which results to be extremely efficient to solve;
- Task augmentation methods, where the redundancy is reduced or eliminated by augmenting the original task with other $s \leq n - m$ auxiliary tasks.

In this thesis the Null-space redundancy resolution is exploited for achieving the proposed objectives.

The Null-space methods make use also of the Jacobian-based solution. Hence, the analysis of this method is firstly proposed.

The goal for Jacobian-based methods is to find a not naive solution for $\dot{r} = J(q)\dot{q}$, for a given configuration q and a given Jacobian J of dimension $m \times n$ with $n > m$. This solution can be expressed as

$$\dot{q} = K(q)\dot{r} \tag{2.7}$$

where $K(q)$ is a $n \times m$ matrix and $\dot{q}$ a m-dimensional vector. Of course if $\dot{r} = 0$, also $\dot{q} = 0$, but in general $\dot{r} \neq 0$. Moreover, there are some minimum requirement on the choice of the $K(q)$ matrix, i.e. K must be a generalized inverse of J, or, mathematically speaking, K must be such that

$$J(q)K(q)J(q) = J(q) \tag{2.8}$$

The most popular choice for the K matrix is the pseudo-inverse of J. Indeed, it always exists, it is unique and of course satisfies the minimum requirement expressed in Equation 2.8. If J is full row rank, then the pseudo-inverse of J can be expressed as

$$J^{\#} = J^T(JJ^T)^{-1} \tag{2.9}$$

and, if the rank condition does not hold, it can still be computed in a numerical way by using the Singular Value Decomposition of J. In particular, the pseudo-inverse of J has the following properties

- $JJ^{\#}J = J$
- $J^{\#}JJ^{\#} = J^{\#}$
- $(JJ^{\#})^T = JJ^{\#}$
- $(J^{\#}J)^T = J^{\#}J$

Moreover, the pseudo-inverse minimizes the joint velocity quadratic norm $\|\dot{q}\|^2 = \dot{q}^T\dot{q}$ among all joint velocities minimizing the task error norm $\|\dot{r} - J(q)\dot{q}\|^2$. Finally, if the task is feasible, i.e. $\dot{r} \in R(J(q))$ the task error will be zero.
Instead of the pseudo-inversion, one may choose also a weighted version of it, i.e.

$$J^{\#}_W = W^{-1}J^T(JW^{-1}J^T)^{-1} \tag{2.10}$$

where W is a weight matrix. This can be used to minimize a weighted norm; indeed, the larger the weight, the smaller will be the joint velocity associated to that weight. However, it should be noticed that this is not a pseudo-inverse anymore, since the fourth property among the ones presented before does not hold.

Another popular choice is the so-called Damped Least Squares (DLS), for its ability of ensuring robustness in presence of singularities. Indeed, when the robot gets close to a singularity, and then its Jacobian is about to lose rank, the pseudo-inversion solution will have a sudden discontinuity. It would be preferable to smooth this solution in order to avoid an increase of the norm velocities due to the joint velocities generated by the pseudo-inversion.

The DLS solution is found by minimizing the following objective function w.r.t. the joint velocities $\dot{q}$

$$H(\dot{q}) = \frac{\mu^2}{2}\|\dot{q}\|^2 + \frac{1}{2}\|\dot{r} - J\dot{q}\|^2 \tag{2.11}$$

and its closed form solution is

$$\dot{q} = J_{DLS}(q)\dot{r} = J^T(JJ^T + \mu^2 I_M)^{-1}\dot{r} \tag{2.12}$$

Basically, it makes a compromise between a large joint velocity and task accuracy. Indeed, it induces a robust behavior when approaching and crossing singularities, paying the presence of a task error as a price, since this basic version is like distributing some damping effects uniformly on each joint.
In particular, the robust behavior is introduced by adding the damping factor μ, since there is no need for caring about the rank of J anymore, because the matrix that will be inverted will always be a positive definite matrix, guaranteeing for the inverse matrix existence. Of course, the larger the damping factor μ, the larger the task error generated by this approach, which will have the following dynamics:

$$\dot{e} = \mu^2(JJ^T + \mu^2 I_m)^{-1}\dot{r} \tag{2.13}$$

In other words, when the robot is far from singularities, it would be better to switch off the μ factor.
Finally, it is worth to notice that J_{DLS} is not a generalized inverse of the robot Jacobian J.

However, the Jacobian-based methods here proposed to handle redundancy have some limitations. In particular, they lead to non-repeatability of cyclic task trajectory: suppose to have a circular trajectory and an initial configuration q_i; after the robot completes the first cycle, the final configuration q_f will be typically different from the initial one. After a second tour, it would change again, and so on. This is due to some drift phenomena introduces by the use of the pseudo-inverse or any other matrix which is at least a generalized inverse of the robot Jacobian. In other words, Jacobian-based methods imply a lack of repeatability. For this work, this could be crucial, since, as deeply discussed in the following chapters, even on cyclic tasks, the robot would be almost always in different configurations, implying an increase in the prediction variance.

As previously anticipated, a term can be added to the solution of the Jacobian-based methods so as to not affect the execution of the task trajectory. This is the main feature characterizing the Null-space methods for handling redundancy, where the self-motion capability of the structure is explored.
Indeed, general solution for $J\dot{q} = \dot{r}$ can be written as

$$\dot{q} = J^{\#}\dot{r} + (I - J^{\#}J)\dot{q}_0 \tag{2.14}$$

where $\dot{q}_0$ is any preferred joint velocity to add in the null-space of J and the term $(I - J^{\#}J)$, also called Projector, is an orthogonal projection matrix that takes any $\dot{q}_0$ and projects it into the null-space of the robot Jacobian.
The properties of the projector matrix are listed as follows:

- it is symmetric, i.e. $(I - J^{\#}J) = (I - J^{\#}J)^T$;
- it is idempotent, i.e. $(I - J^{\#}J)^i = (I - J^{\#}J) \qquad \forall i \in \mathbb{N}$;
- it holds $(I - J^{\#}J)^{\#} = (I - J^{\#}J)$;
- $(I - J^{\#}J)\dot{q}_0$ is orthogonal to $J^{\#}\dot{r}$, meaning that their scalar product is equal to zero.

At this point, it is reasonable to wonder how to choose $\dot{q}_0$. First of all, since the pseudo-inverse solution represents the minimum-norm solution in terms of velocity, it is worth to notice that including a preferred velocity in null-space implies also an increase of the total velocity norm.
Summarizing, a correction is added to the original pseudo-inverse, i.e. minimum norm, solution which is in the null-space of the Jacobian and possibly satisfies additional criteria or objectives.

This framework for handling redundancy is defined at the first differential level, i.e. it is like assuming to command joint velocities, but it can be easily extended to the accelerations level, i.e. the second differential one. It is worth to notice also that torques and accelerations are related to the same differential level, and in particular through the dynamic model.
Working at the acceleration level guarantees some advantages, e.g. in this way discontinuities may happen at the acceleration level, still guaranteeing a smooth velocity behavior, that is actually what physically matters. In other words, discontinuities in the acceleration are still physically feasible. Moreover, it opens the way to a larger set of optimization problems, e.g. the minimization of the torque delivered by the motors for performing a task the robot is redundant with respect to.
To explore the redundancy resolution at the acceleration level, let's start from the definition of a generic task

$$r = f(q) \qquad \longrightarrow \qquad \dot{r} = J(q)\dot{q} \qquad \longrightarrow \qquad \ddot{r} = J(q)\ddot{q} + \dot{J}(q)\dot{q} \tag{2.15}$$

where $\dot{J}(q)\dot{q}$ typically contains quadratic terms in the velocity related to centripetal and Coriolis velocities.
This relation can be rewritten as

$$J(q)\ddot{q} = \ddot{r} - \dot{J}(q)\dot{q} = \ddot{x} \tag{2.16}$$

that is formally equivalent to the first order level, but with acceleration in place of velocity. For instance, the minimum acceleration norm solution in the null-space method can be expressed as

$$\ddot{q} = J^{\#}\ddot{x} + (I - J^{\#}(q)J(q))\ddot{q}_0 \tag{2.17}$$

where $\ddot{q}_0$ represents a preferred acceleration aimed at optimizing some objective function H in the null-space. However, it is important to underline that, while with control schemes at the first order the preferred velocity can be obtained just optimizing some objective function H, here it is also needed to add a damping term aimed at stabilizing the self motions in the null-space, e.g.

$$\ddot{q}_0 = \nabla_q H - K_D \dot{q} \qquad K_D > 0 \tag{2.18}$$

Typically, at the acceleration level the dynamic model of a robot manipulator is used to formulate and solve dynamic problems in the LQ optimization framework, where, assuming a full rank Jacobian, there exist a closed-form solution. The most popular dynamic objective to be locally minimized is the torque norm, exploited with many weighted variants, e.g. weighting the torque norm by the inverse of the inertia matrix. However, in this work, an objective function aimed at minimizing the prediction variance in the LQ framework will be proposed.

2.3 Robot Dynamics

The dynamic model of a robot manipulator exploits the relationship between the generalized force acting on the robot and the motion of the robot itself, i.e. the joint configurations assumed over time.
Assume to have a N DoFs fully-actuated manipulator, whose structure is an open kinematic chain, i.e. without self loops. Then, its dynamic model can be expressed as a system of second order differential equations

$$M(q)\ddot{q} + S(q, \dot{q})\dot{q} + g(q) = \tau \tag{2.19}$$

where, q, $\dot{q}$, $\ddot{q} \in \mathbb{R}^N$ represents the joint positions, velocities and acceleration respectively; $M(q) \in \mathbb{R}^{N\times N}$ represents the robot inertia matrix, guaranteed to be positive definite and symmetric, $S(q, \dot{q})\dot{q} \in \mathbb{R}^N$ the Coriolis and centrifugal terms and $g(q)$ the gravity torques vector. Finally, $\tau \in \mathbb{R}^N$ represents the generalized joint torques vector.
Also, other terms like frictions, if identified, can be easily included in the left side of the dynamic model.
The knowledge of such a model plays an important role for the simulation of the motion of the robot and the design of suitable control laws, allowing to the test for them without the use of a physical system.
Basically, one may distinguish two kinds of dynamic:

- Direct dynamics, where given an input for the robot and its initial state, the resulting motion is directly computed;
- Inverse dynamics, where given a desired motion, the required input to execute that particular motion must be computed.

A very common approach for deriving a dynamic model is the Euler-Lagrange method, an energy based approach that allows to specify the dynamic equations in symbolic form and it is the best for studying and analyzing the dynamic properties and the control schemes.
Other approaches can be viable, like the Newton-Euler one, a recursive method based on the balance of forces and torques, particularly efficient for implementing

inverse dynamics in real time, or others like the principle of d'Alembert, of Hamilton, of virtual works, and so forth.
However, the most common Euler-Lagrange approach is here discussed. It relies into a basic assumption: the robot links in motion are considered as rigid bodies. However, with some modification also some concentrated elasticity at the joints can be included with this method.
Define as $q \in \mathbf{R}^n$, where n is the number of the robot joints, the selected generalized coordinates, e.g. the joint variables. Then, the Lagrangian can be defined as

$$L(q, \dot{q}) = T(q, \dot{q}) - U(q) \tag{2.20}$$

where $T(q, \dot{q})$ and $U(q)$ are given by the sum of the kinetic and potential energy of all joints respectively.
Following, by the Hamilton principle of least action, i.e. during motion the Lagrangian is minimized, and the principle of virtual works, that takes into account the presence of non conservative forces, the Euler-Lagrange equations are derived as follows

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = u_i \qquad i = 1, ..., n \tag{2.21}$$

where u_i represents all the non-conservative forces, i.e. forces changing the total energy of the system, like motor torques or friction. In particular, u_i will be zero if there are no external forces or dissipation of energy. About that, It is important to notice that the gravity force is not an external force, since it is intrinsic to the system. Indeed, it comes from the potential energy.

The robot kinetic energy is defined as the sum of the kinetic energy of each link, i.e.

$$T = \sum_{i=1}^{n} T_i \tag{2.22}$$

Moreover, by considering the robot as an open kinematic chain, the kinetic energy of a generic link i will depend also on the configuration and the velocity of the joints j such that $j \leq i$, i.e. all the previous ones.
Following, by König theorem, the kinetic energy of each link can be expressed as

$$T_i = \frac{1}{2} m_i v_{ci}^T v_{ci} + \frac{1}{2} \omega_i^T I_{ci} \omega_i \tag{2.23}$$

where the first term represents the absolute velocity of the center of mass (CoM) and the second one the absolute angular velocity of the whole body. Also, ω_i and I_{ci} should be expressed in the same reference frame, while the product $\omega_i^T I_{ci} \omega_i$ is invariant with respect to any chosen frame.
Hence, the final expression of T can be written as

$$T = \frac{1}{2} \sum_{i=1}^{n} (m_i v_{ci}^T v_{ci} + \omega_i^T I_{ci}) \tag{2.24}$$

and, collecting quadratic terms with respect to velocity,

$$T = \frac{1}{2} \dot{q}^T M(q) \dot{q} \tag{2.25}$$

where $M(q)$ represents the robot generalized inertia matrix, which is guaranteed to be symmetric and positive definite, i.e. always invertible.

About the potential energy, instead, assume the gravity to be the only contribution; then, the robot potential energy can be expressed as the sum of the potential energy of each link, i.e.

$$U = \sum_{i=1}^{n} U_i \tag{2.26}$$

Again, being the robot an open kinematic chain, the potential energy of the i-th link will depend also on the configuration of the previous joints. Then, the potential energy of a generic link can be computed as

$$U_i = -m_i g^T r_{0,ci} \tag{2.27}$$

where m_i represents the mass of link i, g^T the gravity acceleration vector and $r_{0,ci}$ the position of the center of mass of link i expressed in the base reference frame RF_0.
It is interesting to notice how, for deriving the dynamic model, it is not necessary to build up the Lagrangian and apply the Euler-Lagrange equations. Indeed, just the result of the a of the robot kinetic and potential energies will be used in order to get the full expression of the dynamic model. However, by applying the Euler-Lagrange equations, as a result there will be all linear terms in the acceleration $\ddot{q}$, quadratic terms in the velocity $\dot{q}$ and nonlinear, more in particular trigonometrical, terms in the configuration q.

The final full formulation of the dynamic model, indeed, takes the following form

$$M(q)\ddot{q} + c(q,\dot{q}) + g(q) = u \tag{2.28}$$

where

- the inertia matrix $M(q)$ is obtained by computing the hessian matrix with respect to $\dot{q}$ of the kinetic energy $T(q,\dot{q})$;
- the centrifugal and Coriolis terms $c(q,\dot{q})$ can be computed as

$$\begin{cases} c_k(q,\dot{q}) = \dot{q}^T C_k(q)\dot{q} \\ C_k(q) = \frac{1}{2}\left(\frac{\partial M_k}{\partial q} + \left(\frac{\partial M_k}{\partial q}\right)^T - \frac{\partial M}{\partial q_k}\right) \end{cases} \tag{2.29}$$

  where $c_k(q,\dot{q})$ represents the k-th element of the centrifugal and Coriolis vector $c(q,\dot{q})$ and M_k represents the k-th column of the inertia matrix $M(q)$. This vector can also be factorized as $c(q,\dot{q} = S(q,\dot{q})\dot{q}$ such that $\dot{M} - 2S$ to be skew-symmetric;
- the gravity vector $g(q)$ can be computed as

$$g_k(q) = \frac{\partial U}{\partial q_k} \tag{2.30}$$

  where $g_k(q)$ represents the k-th component of the gravity vector $g(q)$.

To conclude the discussion on the robot dynamics, it is interesting to point out the energy conservation property.
Consider the total robot energy, i.e.

$$E = T + U = \frac{1}{2}\dot{q}^T M(q)\dot{q} + U(q) \tag{2.31}$$

and its dynamics over time, i.e.

$$\begin{aligned}
\dot{E} &= \dot{q}^T M(q)\ddot{q} + \frac{1}{2}\dot{q}^T \dot{M}(q)\dot{q} + U(q) \\
&= \dot{q}^T (u - S(q,\dot{q})\dot{q} - g(q)) + \frac{1}{2}\dot{q}^T \dot{M}(q)\dot{q} + \dot{q}^T g(q) \\
&= \dot{q}^T u + \frac{1}{2}\dot{q}^T \left(\dot{M}(q) - 2S(q,\dot{q})\right)\dot{q}
\end{aligned}$$

If $u \equiv 0$, the total energy is constant, i.e. there is no dissipation or increase, and it results

$$\dot{E} = 0 \qquad \longrightarrow \qquad \dot{q}^T \left(\dot{M}(q) - 2S(q,\dot{q})\right)\dot{q} = 0 \quad \forall q, \dot{q} \tag{2.32}$$

It should be noticed that here any factorization S of the Coriolis and centrifugal terms can be used; indeed, even if the skew-symmetric property does not hold, as the external vector in the quadratic form is the same appearing also inside $\dot{M}$ and S, the whole quadratic form results to be zero.
It can be concluded that, in general, assuming $U \neq 0$ the variation of the total energy is equal to the work of non-conservative forces, i.e.

$$\dot{E} = \dot{q}^T u \tag{2.33}$$

2.4 Robot Control

By robot control it is usually intended to give the right input for successfully completing a given task, accurately executing a motion trajectory or zeroing a positioning error. There are many control schemes that have been developed in the literature. However, one of the most popular is the Feedback Linearization control law.
In nominal conditions, if the initial state is matched, i.e.

$$q(0) = q_d(0) \qquad \dot{q}(0) = \dot{q}_d(0) \tag{2.34}$$

an inverse dynamics control can be performed by applying just a feedforward torque. In particular, given the robot dynamic model in Equation (2.28) and a twice-differentiable reference trajectory for $t \in [0, T]$

$$q_d(t) \qquad \longrightarrow \qquad \dot{q}_d(t), \quad \ddot{q}_d(t) \tag{2.35}$$

applying the feedforward torque

$$u_d = M(q_d)\ddot{q}_d + \eta(q_d, \dot{q}_d) \tag{2.36}$$

where the compact notation $\eta(q_d, \dot{q}_d)$ represents the sum of the Coriolis and centrifugal vector and the gravity vector, yields exact reproduction of the desired motion.

However, in practice there can be a lot of differences with respect to the nominal conditions. Indeed, the initial state could be not matched to the desired trajectory; there can be disturbances on actuators, unknown values in the carried payload, the presence of unmodeled dynamics and so forth. To make the control low more robust to these sources of uncertainty, a feedback is introduced.
It is important to point out that, if the robot is redundant, to take advantage of the redundancy, the trajectory must be defined in the Cartesian space, and not directly in the joint space. There will be an infinite ways for realizing that particular motion, and any of the techniques presented in the previous sections can be applied in order to generate the corresponding joint trajectory. However, the feedback scheme for recovering position and velocity errors should be introduced at the Cartesian level, since the robot must be driven by the Cartesian error.
This can be formulated as follows. Given the dynamic model of a robot manipulator expressed in Equation (2.28) and a twice-differentiable Cartesian desired task

$$p_d(t) \qquad \longrightarrow \qquad \dot{p}_d(t), \quad \ddot{p}_d(t) \tag{2.37}$$

and introducing a feedback scheme at the Cartesian level

$$e_p = K_P(p - p_d) \qquad\qquad \dot{e}_p = K_D(\dot{p} - \dot{p}_d) \tag{2.38}$$

where e_p and $\dot{e}_p$ represent the Cartesian end-effector position and velocity errors respectively, by using any of the redundancy resolution methods discussed before for computing the task reference acceleration $\ddot{q}_{ref}$ in the joint space, the resulting nominal control law will be given by

$$u = \hat{M}(q)\ddot{q}_{ref} + \hat{\eta}(q, \dot{q}) \tag{2.39}$$

This kind of control law is made up by a feedback linearization term, a feedforward term and a PD term for recovering the Cartesian error, and it takes the name of Computed Torque.
In nominal conditions, i.e. $\hat{M} = M$ and $\hat{\eta} = \eta$, applying the control law expressed in Equation (2.39) leads to a linear and decoupled second order system, i.e.

$$\ddot{q} = \ddot{q}_{ref} \tag{2.40}$$

In other words, under a nonlinear controller leading to a perfect feedback linearization, the robot has a dynamic behavior that is invariant, linear and decoupled in its whole state space. In particular, the linearity guarantees for the error transient to exponentially vanish, while the decoupling ensures each joint coordinate to evolve independently from the other, forced by the respective component of $\ddot{q}_{ref}$.
As a remark, the exact linearization by state feedback is a general technique of the nonlinear control theory, and it can be applied to any nonlinear system, e.g. manipulators with elastic joints, wheeled mobile robots, satellites, and so forth.
The control law presented here will also be the model-based baseline for the proposed approach.

2.5 Uncertain Dynamics

In this section the consequences of having an uncertain dynamics will be addressed. Consider a robot manipulator with n degrees of freedom, then, as seen in previous sections, its dynamics is represented by

$$M(q)\ddot{q} + \eta(q,\dot{q}) = \tau \tag{2.41}$$

where $\eta \in \mathbf{R}^n$ is a compact notation representing the sum of the Coriolis and centrifugal vector $c(q,\dot{q})$ and the gravity vector $g(q)$. By assuming the robot to be fully actuated, a Feedback Linearization controller can be designed.
As seen before, in an uncertainty-free case, it provides an input torque τ_{FL} able to cancel all the nonlinear dynamic terms in Equation (2.41), i.e.

$$\tau_{FL} = M(q)u + \eta(q,\dot{q}) \tag{2.42}$$

where $u \in \mathbf{R}^n$ represents the desired joint accelerations. Under this control law, having a perfect knowledge of the dynamic model of the robot, the acceleration appears as linear, i.e.

$$\ddot{q} = u \tag{2.43}$$

However, this can typically not occur because of uncertainties in the dynamic model, like on the links masses, unmodeled dynamics or something like the introduction of a payload at the end-effector level. Taking into account these uncertainties, the dynamic model components can be expressed as

$$M(q) = \hat{M}(q) + \Delta M(q) \tag{2.44}$$

$$\eta(q,\dot{q}) = \hat{\eta}(q,\dot{q}) + \Delta\eta(q,\dot{q}) + \tau_f(\dot{q})) \tag{2.45}$$

where $\hat{M}$ and $\hat{\eta}$ represent the nominal components of the dynamic model, i.e. the ones assumed to be known, e.g. by previous estimation, ΔM and $\Delta\eta$ characterize instead the uncertainties and τ_f represents the joint friction torques.
By applying the nominal Feedback Linearization controller, i.e.

$$\hat{\tau}_{FL} = \hat{M}(q)u + \hat{\eta}(q,\dot{q}) \tag{2.46}$$

on the real system expressed in Equation (2.19), and taking into account also Equations (2.44) and (2.45), it results

$$\ddot{q} = M(q)^{-1}\hat{M}(q)u - M(q)^{-1}(\Delta\eta(q,\dot{q}) + \tau_f(\dot{q})) = u + \varepsilon(q,\dot{q},u) \tag{2.47}$$

where $\varepsilon(q,\dot{q},u)$ represents all the unmodeled dynamic terms.
The main goal of the proposed approach is to provide an estimate of this value $\varepsilon(q,\dot{q},u)$.

Chapter 3

The Proposed Approach

In this chapter, a detailed explanation of the proposed approach is provided.
In particular, by exploiting some machine learning models, the dynamic model uncertainties and parameters variations can be reconstructed.
Indeed, in this thesis an on-line algorithm based on Gaussian Process Regression is addressed. However, any other techniques, such as Neural Networks, Linear Regression or Regression Support Vector Machine could have been successfully employed without any changes on the framework structure.
Anyway, the choice of a regressor based on Gaussian Processes is linked to the property of GPs of providing accurate predictions even with very small datasets, guaranteeing the possibility of an on-line approach.
Another advantage provided by the Gaussian Processes is related to the information about the variance of the estimation, explicitly representing a measure of the estimation error. Moreover, by linearizing the Gaussian Process, this variance information can be also written as a quadratic form in terms of the current acceleration, leading the way to a lot more of possibility. Most likely, being the robot redundant with respect to the proposed desired tasks, this redundancy can be exploited for minimizing the prediction variance, allowing for more accurate predictions.

However, as deeply investigated in the following sections, the standard Gaussian Process regression scales cubically with respect to the number of samples available in the dataset, making the whole method computationally unfeasible for real-time motions. Consequently, the implementation of a sparse approach with some strategy for choosing a compact subset of the dataset representing at best the dataset itself is strongly recommended.
Finally, before getting into the details of the proposed approach, some theoretical notions about Gaussian Process regression and how to get an accurate and computationally efficient linearized version of it are proposed.

3.1 Gaussian Processes

In general, a supervised learning model consists into, given a dataset D of N observations

$$D = \{(x_i, y_i) | i = 1, ..., N\} \tag{3.1}$$

making predictions for new inputs X' not belonging to the current training set. Following, the training phase results to be inductive, since it aims to learn a function f which makes predictions for all possible input values from a finite training dataset D.

Basically, there are two common approaches. The first corresponds to restrict the class of considered functions, e.g. by restricting the infinite set to only a set of functions which are linear with respect to the input. The second, instead, aims at giving a prior probability ti each possible function, where the highest probabilities are given to those functions that are considered to be more likely according to some features, for example one may consider a function more likely with respect to another one because the former results smoother than the latter.
The first approach can lead to poor performance because of the bad modeling of the target function. Indeed, a linear model may not match an highly nonlinear function, scoring bad performance. On the other hand, increasing the flexibility of the class of functions, there is a possibility of getting into the danger of overfitting on the training data, performing well on the already seen samples, but badly predicting for unseen data.
The second approach, instead, highlights the problem of learning the target function among an infinite number of functions in finite time. To solve for this, the Gaussian Processes are introduced.

A Gaussian process is a generalization of the Gaussian probability distribution. Indeed, as a probability distribution is able to characterize scalar or multi-dimensional random variables, a stochastic model will be able to characterize the properties of a function.
As a consequence, the previous problem can be solved by specifying a desired set of features of the function at a finite number of points; then, the inference in the Gaussian process will provide an output based on a restricted set of points, that would be the same as computed by taking into account the whole set.
However, the Gaussian process can be identified as a non-parametric model, i.e. the larger the dataset, the less the posterior uncertainty close to the observations, and the mean function fits itself to be consistent with the dataset samples. However, even with large datasets, there is still some flexibility in the choice of functions.
Another characteristic that can be exploited is the choice of the covariance function. Indeed, by a proper selection of it, the properties that the target function should have are specified. In some sense, the learning problem for Gaussian processes can be summarized as the problem of finding the best covariance function. Indeed, the covariance function itself represents a model of the data, together with properties like smoothness or characteristic length-scale which can be interpreted, playing a key role in the definition of a Gaussian process.

Gaussian processes can be used for both classification problems and regression problems. However, in this thesis, the focus will be directed to the seconds, i.e. dealing with the prediction of continuous quantities.
Formally, a Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.

One may completely characterize a Gaussian process by specifying its mean function

$$m(x) = \mathbf{E}[f(x)] \tag{3.2}$$

and the covariance function

$$k(x, x') = \mathbf{E}[(f(x) - m(x))(f(x') - m(x'))] \tag{3.3}$$

of a real process expressed as

$$f(x) \sim GP(m(x), k(x, x')) \tag{3.4}$$

For notational simplicity and without loss of generality, the mean function can be considered as zero. Moreover, the random variable corresponding to the entry (x_i, y_i) is denoted as $f_i = f(x_i)$. Following, the Gaussian process is defined as a collection of these random variables, implying also the consistency requirement corresponding to the marginalization property. Indeed, considering a larger set of variables does not modify the distribution over the smaller sets.

3.1.1 Kernel Covariance

There are a lot of possibilities for choosing the covariance function. This function, in particular, takes the name of kernel $k(\cdot, \cdot)$. Just to mention, the most popular kernels are the Squared Exponential function and the Matern function; however, one may define his own kernel function in order to representing at best the available data.
In this thesis and in particular, in the proposed approach, a modified Squared Exponential (SE) kernel has been selected. Indeed, to the classic SE kernel, expressed as

$$k(x_i, x_j) = \mathrm{e}^{-\frac{1}{2}\frac{(x_i - x_j)(x_i - x_j)^T}{2\sigma_L}} \tag{3.5}$$

has been pre-multiplied a separate length scale per predictor, denoted as σ_F. This kernel takes the name of ARD SE, i.e. Automatic Relevance Determination Squared Exponential kernel, and it is represented by

$$k(x_i, x_j) = \sigma_F^2 \cdot \mathrm{e}^{-\frac{1}{2}(x_i - x_j)^T \Sigma_L^{-1}(x_i - x_j)} \tag{3.6}$$

where σ_F represents the signal standard deviation, σ_L the characteristic length scale and Σ_L represents the diagonal matrix containing the hyper-parameters σ_L on the main diagonal. Also, the information provided by σ_L can be seen as the distance needed in the input space for having a function value changing significantly.
It is worth to notice that this particular covariance function is infinitely differentiable, guaranteeing for the smoothness of the solution.

3.1.2 GP Predictions

Typically, in order to have a model the more realistic as possible, it is assumed to have only access to a noisy version of the function values, i.e. one may only have knowledge about

$$y = f(x) + \varepsilon \tag{3.7}$$

where ε represents an additive identically distributed Gaussian noise, whose variance is denoted as σ_n^2.
Under this assumption, the prior on the noisy observations is described as

$$\mathrm{cov}(y_i, y_j) = k(x_i, x_j) + \sigma_n^2 \delta_{ij} \tag{3.8}$$

where δ_{ij} is defined as the Kronecker delta

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

Equivalently, by the assumption on the noise to be independent and identically distributed, the covariance for the whole observation set y becomes

$$\mathrm{cov}(y) = K(X, X) + \sigma_n^2 I \tag{3.10}$$

where I represents the identity matrix of proper dimensions.
Following, the joint distribution of the observed target values and the function values at the query points X_* under the prior results to be

$$\begin{pmatrix} y \\ f_* \end{pmatrix} \sim N\left(0, \begin{pmatrix} K(X,X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{pmatrix}\right) \tag{3.11}$$

Hence, by deriving the conditional distribution, the Gaussian process prediction results to be fully described by its mean

$$\mu_* = K(X_*, X)\left(K(X,X) + \sigma_n^2 I\right)^{-1} y \tag{3.12}$$

and its variance

$$\sigma_*^2 = K(X_*, X_*) - K(X_*, X)\left(K(X,X) + \sigma_n^2 I\right)^{-1} K(X, X_*) \tag{3.13}$$

If the samples mean $m(x)$ is different from zero, instead, it is enough to rewrite Equation (3.12) as

$$\mu_* = m(X_*) + K(X_*, X)\left(K(X,X) + \sigma_n^2 I\right)^{-1} y \tag{3.14}$$

Moreover, it is worth to notice how the prediction mean, in Equation (3.12) results to be nonlinear in the input X, but still linear in the observations y! Indeed, one may see it as a linear combination of kernel functions centered on a query point, i.e.

$$\mu_* = \sum_i \alpha_i k(x_i, x_*) \tag{3.15}$$

where

$$\alpha = (K + \sigma_n^2 I)^{-1} y \tag{3.16}$$

It should be noticed also that the variance expressed in Equation (3.13) does not depend on the observations y, but only on the inputs X and X_*. In particular, it is given by the difference between the prior covariance, i.e.

$$K(X_*, X_*) \tag{3.17}$$

and a positive term representing the information about the target function provided by the observations, i.e.

$$K(X_*, X)\left(K(X,X) + \sigma_n^2 I\right)^{-1} K(X, X_*) \tag{3.18}$$

Last but not least, it is important to remark the computational complexity of this algorithm. Indeed, as anticipated in the previous section, the Gaussian process computational cost scales cubically with respect to the number of samples in the dataset. This is mainly due to the matrix inversions required by the equations characterizing the model predictions. However, one may use the Cholesky decomposition, consisting into a factorization of an Hermitian, positive-definite matrix into a product of a lower triangular matrix and its conjugate transpose, instead of the direct inversion of the matrix, in order to speed up the computations and also make the inversion more stable from a numeric point of view.
Anyway, the computational complexity can be solved by implementing a sparse Gaussian process regression model. This basically consists into choosing an active set of the full training dataset and using only the selected subset for making predictions. There are several ways for choosing the active set which can represent at best the whole dataset. Just to mention, some of them are based on the differential entropy, on the log likelihood function, or, as the one used in this thesis, on the approximation of sparse matrices.
However, even if this method significantly reduce the time required for the computations, it still depends in some way on the dimension of the dataset, since an active selection should be done for choosing the active set. Hence, an additive speed up can be obtained by selecting the active set randomly, or just by choosing the last $m < n$ dataset pairs, where n represents the total number of entries of the dataset.

3.1.3 Hyper-Parameters Fitting

So far, the Gaussian process has been presented as a non-parametric model. However, there are some free parameters that, with a proper tuning, improves the quality of the predictions. In particular, these parameters are implicitly introduced with the choice of the covariance function.
In this thesis, the considered covariance kernel is the ARD SE, i.e. the one expressed in Equation (3.6). Here there are two hyper-parameters that can be tuned, the scalar σ_F^2, namely the signal standard deviation, and the d-dimensional vector σ_L representing the characteristic length scale of the signal, where d is the dimension of the input. Also, when assuming noisy observations, a third hyper-parameter, σ_n, namely the noise standard deviation, is introduced.
In this context, the tuning of the hyper-parameters can be performed by exploiting, and in particular maximimizing, the likelihood. Following, denoting the covariance matrix for the noisy observations as

$$K_y = K_f + \sigma_n^2 I \tag{3.19}$$

the marginal likelihood results to be

$$\log p(y|X,\theta) = -\frac{1}{2} y^T K_y^{-1} y - \frac{1}{2} \log |K_y| - \frac{n}{2} \log 2\pi \tag{3.20}$$

From Equation (3.20), three terms can be identified. The first term is the only one involving explicitly the observations data; the second one represents the complexity penalty and it depends only on the covariance function; the third term plays the role as a normalization constant. But how this marginal likelihood depends on the hyper-parameters?
About that, it can be proven the fitting data to decrease monotonically with the length-scale, because the model becomes less and less flexible; at the same time, the negative complexity penalty increases with the length-scale, since the model lose complexity when the length-scale grows.
The likelihood expressed in Equation (3.20) results to have a peak close to 1, while rapidly decreasing for length-scales longer than this value. However, in general, with more data, the complexity term tends to be larger, implying an increase of the length-scales.

The goal is to select the hyper-parameters in order to maximize the marginal likelihood expressed in Equation (3.20). To achieve this goal, one may look for the partial derivatives of the marginal likelihood with respect to the hyper-parameters, getting

$$\begin{aligned}\frac{\partial}{\partial\theta_j}\log p(y|X,\theta) &= \frac{1}{2}y^T K^{-1}\frac{\partial K}{\partial\theta_j}K^{-1}y - \frac{1}{2}\mathrm{tr}\left(K^{-1}\frac{\partial K}{\partial\theta_j}\right)\\ &= \frac{1}{2}\mathrm{tr}\left((\alpha\alpha^T - K^{-1})\frac{\partial K}{\partial\theta_j}\right) \qquad \alpha = K^{-1}y\end{aligned} \tag{3.21}$$

As previously investigated, the standard computations require the inversion of the covariance matrix, that is positive definite and symmetric, and then a computational cost scaling cubically with respect to the data. However, once this inverse is known, the computation of the derivatives expressed in Equation (3.21) requires only a quadratic time per hyper-parameter. In other words, the tuning of the hyper-parameters through the gradient based method presented in this section does not grave on the final asymptotic time complexity of the overall algorithm.

3.2 Linearized Gaussian Processes

In the previous sections, the Gaussian processes and the advantages they can bring have been explored. However, the exact Gaussian process may result unpractical for real-time applications due to its cubic computational time. For this purpose, a sparse approach has been proposed, temporarily solving the problem. Unfortunately, this issue may reoccur whenever some other optimization is considered. Indeed, the inference at any input, due to the covariance function, results to be non-convex, making any optimization involving the Gaussian process regression to be non-convex, and then difficult to solve. In general, there are non-linear solvers that can be used, but this can significantly slow down the computational time at each step, falling again in the computational time issue.

In order to have a more reliable real-time applications, some other approach is needed. Typically, in order to overcome this problem, the mean and the variance of the Gaussian process prediction are approximated by linear or quadratic equations by using the first or the second Taylor's series expansion. This basically leads the way to an approximated linear or quadratic problem that can be easily solved. However, this approach may not working well. Indeed, linearizing after the prediction, i.e. directly the mean and the variance of the Gaussian process prediction, the latter may lose the property of a variance, i.e. it may result, due to the Taylor series expansions, into a negative value.

To fix this issue, the linearization will be applied before the Gaussian process prediction.
Consider x to be a given input, and the target function f to be differentiable at x. Also, assume the covariance function $k(\cdot,\cdot)$ to be differentiable. It is worth to notice that this assumption is not restrictive, since the kernel can be arbitrarily chosen. However, the most used ones satisfy this condition, e.g. the ARD Squared Exponential kernel is infinitely differentiable. Then, by recalling the Gaussian process of a target function f to be essentially the posterior distribution over the space of realizations of f, the target function can be linearized around x, getting the following result

$$f(x+\Delta_x) \approx \tilde{f}_x(\Delta_x) = f(x) + \Delta_x^T \nabla_x f(x) \qquad \Delta_x \in \mathbf{R}^n \tag{3.22}$$

Following, by denoting the gradient of the target function f as

$$g(x) = \nabla_x f(x) \tag{3.23}$$

and by defining

$$\hat{x} = \begin{pmatrix} 1 \\ \Delta_x \end{pmatrix} \tag{3.24}$$

the linearization formulated in Equation (3.22) can be rewritten in a compact form as

$$\tilde{f}_x(\Delta_x) = \begin{pmatrix} 1 \\ \Delta_x \end{pmatrix}^T \begin{pmatrix} f(x) \\ g(x) \end{pmatrix} = \hat{x}^T \hat{f}(x) \tag{3.25}$$

In other terms, the original Gaussian process of the target function f has been approximated around the input x by a process of the linearized target function $\tilde{f}$, that take the name of linearized Gaussian Process, or, equivalently, linGP.
As a consequence, $\hat{f}$ will represent a multivariate Gaussian process derived from the original one.
Therefore, the posterior distribution over the values of $f(x)$ and its gradient at the input is defined by the Gaussian random vector $\hat{f}_x$. As a consequence, the distribution of the linearized Gaussian process $\tilde{G}_{f|x}$ is derived from the posterior $\mathbb{P}(f|X,y)$ over the linear approximations of the target function around x.

In order to characterize the linearized Gaussian process, first the Gaussian process of $\hat{f}$ must be defined.
To get into the mathematical formulation, first some notation needs to be introduced. In particular, define

- $K^{(1,0)} = \nabla_x k$ as the gradient of the covariance function k with respect to its first argument;

- $K^{(0,1)} = D_{x'}k$ as the Jacobian of k with respect to its second argument;
- $K^{(1,1)}(x, x')$ as the $n \times n$ matrix got by deriving k w.r.t. both arguments, i.e. such that $K^{(1,1)}_{i,j}(x, x') = \frac{\partial^2}{\partial x_i \partial x'_j} k(x, x')$.

Then, the joint distribution of the observation data, the process f, and the gradient process g can be written as

$$\begin{pmatrix} Y \\ f \\ g \end{pmatrix} \sim N\left(0, \begin{pmatrix} K(X,X) + \sigma_n^2 I & k(X,x) & K^{(0,1)}(X,x) \\ k(x,X) & k(x,x) & K^{(0,1)}(x,x) \\ K^{(1,0)}(x,X) & K^{(1,0)}(x,x) & K^{(1,1)}(x,x) \end{pmatrix}\right) \tag{3.26}$$

where X, x represent the actual dataset and the query point respectively. Moreover, it should be pointed out that whenever a function is applied to the whole dataset X, it gets broadcast along the dimensions of X. In other words, $K(X, X)$ will be a $n \times n$ matrix where the element $(i.j)$ is defined as

$$K_{i,j}(X, X) = k(x^{(i)}, x^{(j)}) \tag{3.27}$$

Following, the Gaussian process of $hatf$ conditioned on the observation data (X, y) will be

$$\mathbf{P}(\hat{f}|X, Y, x) \sim N(\hat{m}_x, \hat{V}_x) \tag{3.28}$$

i.e. a Gaussian with mean

$$\hat{m}_x = \begin{pmatrix} k(x,X) \\ K^{(1,0)}(x,X) \end{pmatrix} (K(X,X) + \sigma_n^2 I)^{-1} Y \tag{3.29}$$

and variance given by

$$\begin{aligned} \hat{V}_x = & \begin{pmatrix} k(x,x) & K^{(0,1)}(x,x) \\ K^{(1,0)}(x,x) & K^{(1,1)}(x,x) \end{pmatrix} \\ & - \begin{pmatrix} k(x,X) \\ K^{(1,0)}(x,X) \end{pmatrix} \left(k(X,X) + \sigma_n^2 I\right)^{-1} \begin{pmatrix} k(X,x) & K^{(0,1)}(X,x) \end{pmatrix} \end{aligned} \tag{3.30}$$

Therefore, the linearized Gaussian process expressed in Equation (3.25) as an approximation of the target function can be written as

$$\mathbf{G}_{f|x} = \tilde{f}_x(\Delta_x) \sim N(\hat{m}_x^T \hat{x},\ \hat{x}^T \hat{V}_x \hat{x}) \tag{3.31}$$

where the mean

$$\tilde{\mu}_{f|x}(\Delta_x) = \hat{m}_x^T \hat{x} \tag{3.32}$$

is a linear function of $\hat{x}$ approximating the original Gaussian process mean at $(x + \Delta_X)$, and the variance

$$\tilde{\sigma}^2_{f|x}(\Delta_x) = \hat{x}^T \hat{V}_x \hat{x} \tag{3.33}$$

results into a quadratic function with respect to $\hat{x}$ approximating the original GP one.

It is important to notice that, being $\hat{f}$ a valid GP, $\hat{V}_x$ necessarily results into a positive semi-definite matrix, guaranteeing for the variance expressed in Equation (3.33) to be non-negative, i.e. preserving the property of the variance.

Moreover, the variance so defined results to be convex, leading the way to the formulation of a quadratic optimization problem, as formalized in the next chapter. In other terms, the linGP is a Gaussian process of a function $\tilde{f}_x$ which approximates the original GP around x.

3.3 Supervised Feedback Learning

Strong of the theoretical and notational background previously discussed, in this section a description of how to apply such notions to a redundant robot is reported. In order to compute the GP regression predictive distribution, a set of n unidimensional Gaussian Processes regressors has been employed and stacked together, where n represents the number of DoFs of the manipulator.

The basic idea relies into the possibility of reconstructing on-line the needed dynamic correction, in terms of joint torques, by comparing the actual motion of the robot with the reference obtained by taking into account the Feedback Linearization prediction.
Assume T_c to be the sampling time, and the robot current state at time $t = t_k = kT_c$ to be defined as

$$x_k = \begin{pmatrix} q_k \\ \dot{q}_k \end{pmatrix} \tag{3.34}$$

Moreover, suppose the desired state at time $t = t_{k+1}$ t0 be

$$x_{d,k+1} = \begin{pmatrix} q_{d,k+1} \\ \dot{q}_{d,k+1} \end{pmatrix} \tag{3.35}$$

Following, by using the nominal Feedback Linearization controller

$$\hat{\tau}_{FL,k} = \hat{M}(q_k)u_k + \hat{\eta}(q_k, \dot{q}_k) \tag{3.36}$$

because of the uncertain dynamics, the robot will not be able to reach the desired state, reaching a different one, defined as

$$x_{k+1} = \begin{pmatrix} q_{k+1} \\ \dot{q}_{k+1} \end{pmatrix} \tag{3.37}$$

Therefore, taking into account the mismatch between the desired robot motion and the performed one, by numerical differentiation, it is possible to reconstruct the robot acceleration $\ddot{q}_k$ and then to estimate the missing torque needed for compensating the unmodeled dynamics. Consequently, a regressor ε can be introduced in the nominal FL control scheme, aimed to progressively compensate the unknown perturbations. In particular, the regressor at time k will be the current robot state and the last commanded acceleration, and the new FL-based control input will be the sum between the nominal one plus the correction term, i.e.

$$\tau_{FL,k} = \hat{\tau}_{FL,k} + \varepsilon_k = \hat{M}(q_k)u_k + \hat{\eta}(q_k, \dot{q}_k) + \varepsilon_k \tag{3.38}$$

3.4 Dataset Generation

As in every supervised learning model, the role of how to build up the dataset is key. Indeed, the choice of the dataset can significantly affect the overall performance scored by the learning approach.
First of all, assume the possibility for the robot to be driven by commanding joint torques. The current state at time $t = k$ of the robot is identified by

$$x_k = (q_k, \dot{q}_k)^T \tag{3.39}$$

i.e. by joints configuration and velocity at time $t = k$, and the goal is to reach a desired target state at $t = k + 1$, i.e.

$$x_{d,k+1} = (q_{d,k+1}, \dot{q}_{d,k+1})^T \tag{3.40}$$

Moreover, the desired input acceleration u driving the robot to the desired state can be computed by the previous techniques of redundancy resolution, like finding the solution by the Jacobian based methods, such as pseudo-inverse or damped least square solutions, and then adding a null Space preferred acceleration optimizing some objective function. In particular, in the proposed approach it has been computed as

$$\ddot{q} = J_{DLS}(q)(\ddot{r} - \dot{J}\dot{q}) = J^T(JJ^T + \mu^2 I_M)^{-1}(\ddot{r} - \dot{J}\dot{q}) \tag{3.41}$$

Following, by applying the nominal Feedback Linearization controller, taking into account the effect of residual and unmodeled dynamics, the robot will reach a different state $x_{k+1} = (q_{k+1}, \dot{q}_{k+1})$.
At this point, it is possible to compute the unmodeled dynamics, which will depend on the actual state and the desired and the actual acceleration at time $t = k$.
By collecting these values for several states, the dataset

$$D = \{(X_i, Y_i) | i = 1, ..., n_d\} \tag{3.42}$$

where n_d represents the number of available elements, can be incrementally built.
In particular, for each sample i, it will result

$$X_i = (q_i, \dot{q}_i, u) \qquad Y_i = u - \ddot{q} \tag{3.43}$$

It is worth to point out that this dataset satisfies the properties of uniqueness and causality, which play an important role for learning the inverse dynamics.
Summarizing, the data collection procedure is based on the mismatch between the commanded joints acceleration and the actual acceleration.

Chapter 4

Variance Optimization

As investigated in the previous sections, there are many ways to handle and take advantage of redundancy. In particular, in this thesis, the remaining degrees of freedom are exploited to minimize the variance of the Gaussian process predictions. This variance, indeed, represents a measure of the prediction uncertainty, and a lower value can lead to an improvement in terms of estimation accuracy, implying better performance.
In other words, a statistical model like the one defined by the Gaussian process, finds itself to be unreliable if the state passes into a region with sparse training data, or, roughly speaking, in states that depart from the region already explored.
As a consequence, the impossibility of reliably predicting the behavior of a nonlinear system when passing into a region with sparse training data represents a critical limitation for the statistical models, making the whole model to be highly dependent on the training data, even if the learning algorithm works perfectly.
Following, the idea relies into choosing as preferred null-space acceleration the one minimizing this variance, in order to lead the robot into states that are near regions where many training data points are available, implying an improvement in terms of accuracy and reliability for the statistical model.
If the robot is redundant, the null-space, i.e. the remaining degrees of freedom, can be exploited while tracking a desired trajectory, driving the robot toward already explored areas through self-motions. The variance, in particular, can be seen as a measure of the reliability of the regions.
To this purpose, taking into account the presence of n Gaussian processes, one for each joint, the function to be minimizing will be the sum of each prediction variance, as investigated in the following sections.

4.1 Nonlinear Optimization

In a nonlinear context, i.e. by using the standard Gaussian processes, the variance is given by Equation (3.13).
Following, the function to be minimized will be given by

$$\sigma^2 = \sum_{i=1}^{n} \sigma_i^2 \tag{4.1}$$

where n represents the number of degrees of freedom of the robot.
As anticipated in the previous chapter, because of the covariance kernel, it results into a non-convex problem, making the use of nonlinear solvers to be required in order to find a solution for the optimization problem.
This necessarily slows down the whole algorithm, making the real-time application of such algorithm very challenging.
However, by using the linearized Gaussian processes, this problem can be reformulated into a convex LQ problem, efficiently solved by a suitable solver.

4.2 Quadratic Programming

The use of the linearized Gaussian process makes the prediction variance to be expressed as in Equation (3.33). Being the goal to minimize the overall uncertainty, and not the individual one on each joint, objective function will be described by

$$\sigma^2 = \hat{x}^T (\sum_{i=1}^{n} V_i) \hat{x} = \hat{x}^T V \hat{x} \tag{4.2}$$

Following, being V positive definite and symmetric, Equation (4.2) can be efficiently solved by quadratic programming.

At this point, there are three possibilities:

- the first, to choose $\ddot{q}_0$ such that the prediction variance get minimized at time t;
- the second, to choose $\ddot{q}_0$ such that the prediction variance get minimized at time $t+1$;
- the third, to choose $\ddot{q}_0$ by taking into account and suitably weighting the prediction variance both at time t and $t+1$.

In this thesis, after several trials, the third case has been selected.

4.2.1 Variance Optimization at Current Time Instant

Consider first the variance at the current time t. Following, $\hat{x}$ at time t can be expressed as

$$\hat{x}_t(\ddot{q}_t) = \begin{pmatrix} 1 \\ \Delta_{x,t} \end{pmatrix} \tag{4.3}$$

where

$$\Delta_{x,t} = \begin{pmatrix} q_t - q_{lin} \\ \dot{q}_t - \dot{q}_{lin} \\ \ddot{q}_t - \ddot{q}_{lin} \end{pmatrix} \tag{4.4}$$

and q_{lin}, $\dot{q}_{lin}$, $\ddot{q}_{lin}$ represent respectively the joint configuration, velocity ad acceleration at the linearization point.

Being q_t and $\dot{q}_t$ fixed, Equation (4.3) can be split into a constant and a time-varying part, i.e.

$$\hat{x}_t(\ddot{q}_t) = b_t + w_t = \begin{pmatrix} 1 \\ q_t - q_{lin} \\ \dot{q}_t - \dot{q}_{lin} \\ -\ddot{q}_{lin} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \ddot{q}_t \end{pmatrix} \tag{4.5}$$

Following, it does hold

$$\begin{aligned} \sigma_t^2 &= (b_t + w_t)^T V (b_t + w_t) \\ &= b_t^T V b_t + w_t^T V b_t + b_t^T V w_t + w_t^T V w_t \end{aligned} \tag{4.6}$$

and, taking into account the symmetry of V,

$$\sigma_t^2 = b_t^T V b_t + 2 b_t^T V w_t + w_t^T V w_t \tag{4.7}$$

Hence, the variance expressed in Equation (4.7) is a complete quadratic problem with respect to w_t. As a recall, the goal was to find $\ddot{q}_t$ such that the variance gets minimized. For this purpose, the constant term, i.e. $b_t^T V b_t$, can be dropped out, and the term w_t can be expressed as

$$w_t = A_t \ddot{q}_t \tag{4.8}$$

where, by assuming n to be the number of degrees of freedom of the manipulator, it results

$$A_t = \begin{pmatrix} 0_{1\times n} \\ 0_{nxn} \\ 0_{nxn} \\ I_{nxn} \end{pmatrix} \tag{4.9}$$

As a result, the proposed quadratic problem with respect to $\ddot{q}_t$ aimed at minimizing the variance at the current time t can be written as

$$\ddot{q}_t = \text{argmin} \left(\frac{1}{2} \ddot{q}_t^T H_t \ddot{q}_t + f_t^T \ddot{q}_t \right) \tag{4.10}$$

where

$$H_t = 2 A_t^T V A_t \qquad\qquad f_t = 2 b_t^T V A_t \tag{4.11}$$

4.2.2 Variance Optimization at Next Time Instant

The next step consists into formulating a complete quadratic problem with respect to $\ddot{q}_t$ aimed at minimizing the variance at time $t+1$.
Following, $\hat{x}$ at time $t+1$ can be written as

$$\hat{x}_{t+1}(\ddot{q}_t) = \begin{pmatrix} 1 \\ \Delta_{x,t+1} \end{pmatrix} \tag{4.12}$$

where

$$\Delta_{x,t+1} = \begin{pmatrix} q_{t+1} - q_{lin} \\ \dot{q}_{t+1} - \dot{q}_{lin} \\ -\ddot{q}_{lin} \end{pmatrix} \tag{4.13}$$

and q_{t+1}, $\dot{q}_{t+1}$, represent respectively the joint configuration, velocity ad acceleration at the next step $t+1$. It is important to notice that the joint acceleration at time $t+1$ has not been taken into account, since the joint motion executed at time $t+1$ is only affected by the joint acceleration provided at time t, then nothing can be said about the joint acceleration at time $t+1$.
However, the quantities q_{t+1} and $\dot{q}_{t+1}$ can be numerically computed as follows. Indeed, by using the Semi-Implicit Euler method, both of them can be expressed as a function of $\ddot{q}_t$, i.e.

$$\begin{cases} \dot{q}_{t+1} & = \dot{q}_t + \ddot{q}_t \cdot \Delta_t \\ q_{t+1} & = q_t + \dot{q}_{t+1} \cdot \Delta_t = q_t + \dot{q}_t \cdot \Delta_t + \ddot{q}_t \cdot \Delta_t^2 \end{cases} \tag{4.14}$$

Following, substituting Equation (4.14) into Equation (4.12), it results

$$\hat{x}_{t+1}(\ddot{q}_t) = \begin{pmatrix} 1 \\ q_t + \dot{q}_t \cdot \Delta_t + \ddot{q}_t \cdot \Delta_t^2 - q_{lin} \\ \dot{q}_t + \ddot{q}_t \cdot \Delta_t - \dot{q}_{lin} \\ -\ddot{q}_{lin} \end{pmatrix} \tag{4.15}$$

Again, being the current joint position and velocity fixed, Equation (4.3) can be split into a constant and a time-varying part, i.e.

$$\hat{x}_{t+1}(\ddot{q}_t) = b_{t+1} + w_{t+1} = \begin{pmatrix} 1 \\ q_t + \dot{q}_t \cdot \Delta_t - q_{lin} \\ \dot{q}_t - \dot{q}_{lin} \\ -\ddot{q}_{lin} \end{pmatrix} + \begin{pmatrix} 0 \\ \ddot{q}_t \cdot \Delta_t^2 \\ \ddot{q}_t \cdot \Delta_t \\ 0 \end{pmatrix} \tag{4.16}$$

Hence, the variance at the next step can be written as

$$\begin{aligned} \sigma_{t+1}^2 &= (b_{t+1} + w_{t+1})^T V (b_{t+1} + w_{t+1}) \\ &= b_{t+1}^T V b_{t+1} + w_{t+1}^T V b_{t+1} + b_{t+1}^T V w_{t+1} + w_{t+1}^T V w_{t+1} \\ &= b_{t+1}^T V b_{t+1} + 2 b_{t+1}^T V w_{t+1} + w_{t+1}^T V w_{t+1} \end{aligned} \tag{4.17}$$

where the symmetric of V has been used.
As in the previous case, the term $b_{t+1}^T V b_{t+1}$ is constant and does not affect the minimization problem, then can be dropped out. Moreover, w_{t+1} can be rewritten as

$$w_{t+1} = A_{t+1} \ddot{q}_t \tag{4.18}$$

where

$$A_{t+1} = \begin{pmatrix} 0_{1 \times n} \\ I_{n \times n} \cdot \Delta_t^2 \\ I_{n \times n} \cdot \Delta_t \\ 0_{n \times n} \end{pmatrix} \tag{4.19}$$

Finally, the quadratic problem aimed at minimizing the variance at the next time step $t+1$ is fully described by

$$\ddot{q}_t = \text{argmin} \left(\frac{1}{2} \ddot{q}_t^T H_{t+1} \ddot{q}_t + f_{t+1}^T \ddot{q}_t \right) \tag{4.20}$$

where

$$H_{t+1} = 2 A_{t+1}^T V A_{t+1} \qquad f_{t+1} = 2 b_{t+1}^T V A_{t+1} \tag{4.21}$$

4.2.3 Complete Quadratic Problem

In the previous sections, both the quadratic problems for the minimization of the variance at time t and $t+1$ have been deeply discussed.
The approach proposed by this thesis aims to take into account both the information, with the addition of a suitable weight to give priority at the current step or at the next one. Indeed, it is worth to notice that both the previous problems share the variable that needs to be optimized, i.e. the joint acceleration $\ddot{q}_t$.
Hence, the final LQ problem can be written as

$$\ddot{q}_t = \text{argmin} \frac{1}{2} \ddot{q}_t^T (\alpha H_t + \beta H_{t+1}) \ddot{q}_t + (\alpha f_t^T + \beta f_{t+1}^T) \ddot{q}_t \tag{4.22}$$

where $\alpha, \beta > 0$ are suitably tuned weights making a trade-off between the minimization of the variance at time t and the one at time $t+1$. This choice empirically makes sense, since the algorithm will try to explore the next instant of time without going too far from the safe zone represented by the reliable area.

Chapter 5

Simulation Results

The simulation is performed on a KUKA LWR 4+ robot, a 7 degrees of freedom manipulator with all revolute joints. A sketch of the considered robot is represented in Figure 5.1. This chapter is organized as follows. First, the model uncertainties and

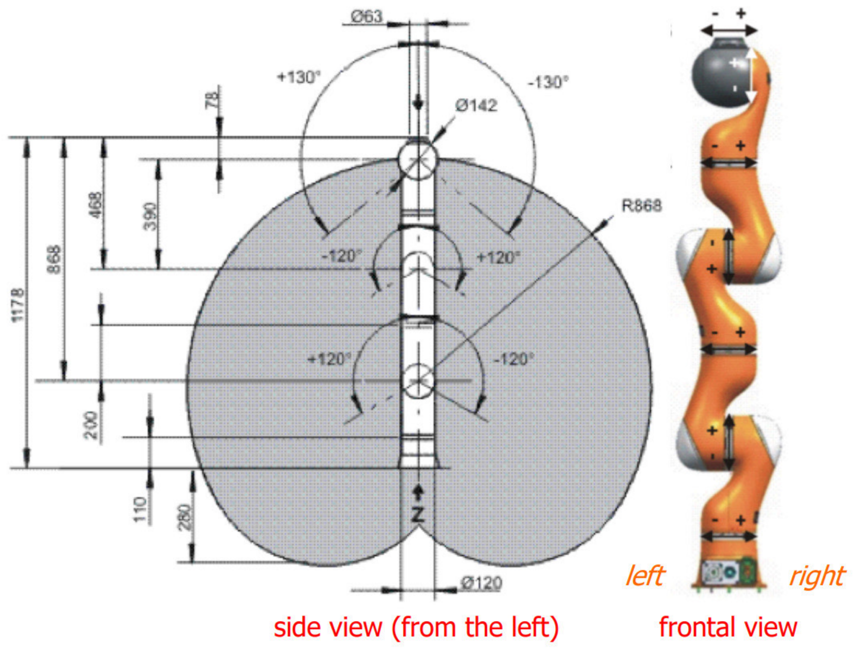


Figure 5.1. Frontal and side views of the KUKA LWR 4+ robot.

the nominal operational settings will be described. Then, the linearized Gaussian process setup used for implementing the learning algorithm and improving the model-based approach is given. Following, a brief section will be dedicated to the definition of the tasks needed for the validation of the proposed algorithm. In particular, the robot will track different trajectories in the Cartesian space, where only the position matters, guaranteeing a 4-dimensional Jacobian null-space.

5.1 Operational Settings

In this section the operational settings used for running the simulations are reported. First of all, the robot initial state is assumed to be matched with the initial desired state, in order to avoid an high error when the motion starts. Once defined the task, this can be easily achieved by solving an inverse kinematic problem for that particular Cartesian point, getting one of the corresponding joint configurations, that, as a recall, can be multiple.

5.1.1 Nominal settings

First, the setup characterizing the nominal control settings is described. About the chosen control law, the damped pseudo-inverse used for computing the DLS solution for handling redundancy is characterized by a damping factor

$$\mu^2 = 0.01 \tag{5.1}$$

Following, the Cartesian PD gains have been chosen as

$$K_P = 100\ I_{3\times 3} \tag{5.2}$$
$$K_D = 2\sqrt{K_P} = 20\ I_{3\times 3} \tag{5.3}$$

where K_P, K_D represent the proportional and the derivative gain respectively and are needed to recover the Cartesian error, while $I_{3\times 3}$ represents the 3×3 identity matrix.
Moreover, the derivative gain K_V specifying the damping term added in the null-space with the aim of stabilizing the self motion when optimizing some objective function at the acceleration level, as in Equation (2.18), is here set as

$$K_V = 10 \tag{5.4}$$

About the optimization itself, instead, a bound on the amount of acceleration that can be provided has been set. In particular, the preferred acceleration must satisfy the following constraints

$$|\ddot{q}_{0,i}| < 10 \qquad \text{rad/sec}^2 \tag{5.5}$$

This bound was mandatory in order to avoid too large variations of the overall joint accelerations that inevitably would make the variance to increase, self-defeating the proposed goal. Also, being in a simulation context, the update of the state of the robot, namely the joint position and the joint velocity, is performed by computing the actual joint acceleration by inverting the real dynamic model, i.e.

$$\ddot{q}_t = M(q_t)^{-1}(\tau_t - \eta(q_t, \dot{q}_t)) \tag{5.6}$$

and then by numeric integration, i.e.

$$q_{t+1} = q_t + \dot{q}_t \Delta_t \tag{5.7}$$
$$\dot{q}_{t+1} = \dot{q}_t + \ddot{q}_t \Delta_t \tag{5.8}$$

Finally, both the simulation and the control steps were set to 10 milliseconds.

5.1.2 Model Uncertainties

In order to simulate the robot and test the proposed algorithm, some uncertainties have been assumed on the masses of the nominal model, corresponding to the one that will be used for controlling the robot through nominal feedback linearization. In particular, the following uncertainties have been assumed on the masses of the links from the first up to the seventh link:

- -60% on the first link mass;
- +40% on the second link mass;
- -50% on the third link mass;
- -30% on the fourth link mass;
- +20% on the fifth link mass;
- -20% on the sixth link mass;
- +20% on the seventh link mass.

On the end-effector, instead, no uncertainty has been assumed. It is important to point out that the uncertainty has been placed on the link masses instead of directly in the dynamic coefficients affecting the dynamic model components in order to guarantee to have still a physically feasible system. Moreover, under some assumption on the distribution of the masses, there is no need to modify also the inertia components. Hence, the results of the simulation for each task are fully reported and discussed, making also a comparison between the standard model-based approach and the one improved with the learning correction presented in this work.

5.1.3 GP Settings

To conclude the operational settings, some hyper-parameters about the proposed learning approach must be selected. In this work, the proposed learning method is represented by the linearized Gaussian process.
In particular, in order to reduce the overall computational complexity, the fitting of the kernel hyper-parameters, namely the signal standard deviation and the signal length-scale, will take place each 30 iterations of the algorithm.
For similar purposes, the linearization point is updated each 5 iterations, providing a significant speed up to the overall algorithm, since it will be forced to make heavier operations, e.g. the ones about kernel computations, only ones each five time steps. Together with the linearization point, also the whole training set, containing all the previous observations, gets updated, stacking the dataset structure and recovering the skipped entries, in order to guarantee for data consistency.
Also, in order to have more accurate predictions, the signal mean has been considered and inserted into the prediction mean equation.
Moreover, to handle long trajectories, a sparse linearized Gaussian process has been implemented, with an active set dimension of $h = 30$ samples. About the selection of the active set, different options have been considered:

- as first, to perform an active selection according to the Sparse Greedy Matrix Approximation (sgma) algorithm, that aims to choose the desired number h of samples able to represent at best the whole dataset;
- following, to select randomly the desired number h of samples among the whole set of observations to build up the training set;
- lastly, to keep only the last h samples of the whole dataset as training set.

The one providing the best performance resulted to be the first. Because of that, this method will be used for the simulations that will follow. However, it was also the slowest.
In order to have a faster selection, i.e. a faster algorithm, one may consider to choose the second or the third method, still providing a good accuracy in terms of predictions and variance. In particular, the third method shone from a local point of view. Indeed, by taking only the last observations, the dataset covariance was locally low, reducing also the prediction uncertainty.

Finally, the ultimate control law, taking into account both the nominal control law and the corrective prediction, can be formulated as

$$\tau = \hat{M}(q)\ddot{q}_{aug} + \hat{\eta}(q, \dot{q}) \tag{5.9}$$

where $\ddot{q}_{aug}$ represents the augmented reference acceleration given by

$$\ddot{q}_{aug} = \ddot{q}_{nom} + \ddot{q}_{pred} \tag{5.10}$$

and $\ddot{q}_{nom}, \ddot{q}_{pred}$ represent the nominal joint acceleration reference and the predicted corrective acceleration respectively.

5.2 Trajectory Planning

In order to test and validate the performance of the proposed approach, some tasks with respect to which the robot is redundant need to be defined.
Consequently, three trajectories have been defined in the Cartesian space, where only the position matters, requiring at least 3 degrees of freedom to be executed:

- a periodic circular trajectory;
- a sinusoidal trajectory;
- a heart-shaped trajectory.

A trajectory is made up by a geometric path and a timing law associated to the path. Basically, there are two ways to plan a trajectory. The first is to define separately the geometric path and the timing law, while the second relies into plan both of them at the same time.
In this thesis, the proposed tasks have been planned according to the first approach, and in particular the timing law will be the same for each trajectory.
Hence, to generate a trajectory, a geometric path $r(s)$ and a timing law $s(\tau)$ are needed.

5.2.1 Timing Law

The timing law specifies the time evolution. It is defined as $s(\tau) \in [0,1]$, where $\tau \in [0,1]$ represents the normalized time

$$\tau = t/T \tag{5.11}$$

and $T = 5$ is the total motion time.
In this thesis, a cubic polynomial function is proposed as timing law, in order to make the movements of the robot smooth, and, under the assumption of making only *rest-to-rest* motions, along with the boundary conditions it results to be expressed as follows:

$$\begin{cases} s(\tau) = a\tau^3 + b\tau^2 + c\tau + d \\ \dot{s}(0) = 0 \quad \rightarrow \quad c = 0 \\ \dot{s}(1) = 0 \quad \rightarrow \quad 3a + 2b = 0 \\ s(0) = 0 \quad \rightarrow \quad d = 0 \\ s(1) = 1 \quad \rightarrow \quad a + b = 1 \end{cases} \tag{5.12}$$

where, substituting to obtain the coefficients explicitly, it results into

$$s(\tau) = -2\tau^3 + 3\tau^2, \tag{5.13}$$

Moreover, differentiating with respect to time, the first and the second derivative can be computed and later exploited for computing the desired velocity and acceleration. In particular, it results

$$\dot{s}(\tau) = \frac{6t}{T^2} - \frac{6t^2}{T^3}, \qquad \ddot{s}(\tau) = \frac{6}{T^2} - \frac{12t}{T^3}. \tag{5.14}$$

5.2.2 Cartesian Periodic Circular Path

First of all, a path can be designed either in the joint or in the Cartesian space, making sure that every point of the path is not outside of the robot workspace. In this work all the proposed path will be planned on the Cartesian space.

The first one is a periodic circular path, defined as

$$p_{ref}(s) = \text{center} + \text{radius} \cdot \begin{pmatrix} \cos(4\pi \cdot s) \\ \sin(4\pi \cdot s) \\ 0 \end{pmatrix} \tag{5.15}$$

around a center defined as

$$\text{center} = \begin{pmatrix} 0.5 \\ 0 \\ 0.1 \end{pmatrix} \text{ m} \tag{5.16}$$

and with a radius

$$\text{radius} = 0.15 \text{ m} \tag{5.17}$$

Following, the desired Cartesian velocity and acceleration are defined by simply differentiating Equation (5.15) with respect to time, getting

$$\dot{p}_{ref} = 4\pi \cdot radius \cdot \begin{pmatrix} -\sin(4\pi \cdot s) \cdot \dot{s} \\ \cos(4\pi \cdot s) \cdot \dot{s} \\ 0 \end{pmatrix} \tag{5.18}$$

$$\ddot{p}_{ff} = 4\pi \cdot radius \cdot \begin{pmatrix} -4\pi \cos(4\pi \cdot s) \cdot \dot{s}^2 - \sin(4\pi \cdot s) \cdot \ddot{s} \\ -4\pi \sin(4\pi \cdot s) \cdot \dot{s}^2 + \cos(4\pi \cdot s) \cdot \ddot{s} \\ 0 \end{pmatrix} \tag{5.19}$$

A graphical representation of this task and the robot initial configuration can be seen in Figure 5.2.

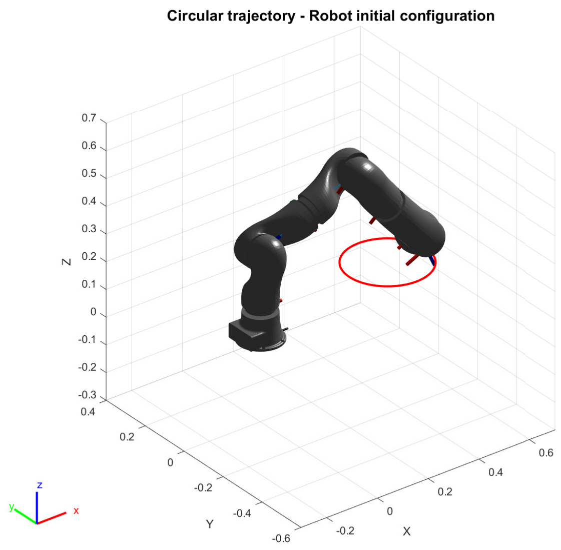


Figure 5.2. Orthographic projection of the Cartesian circular task.

5.2.3 Cartesian Sinusoidal Path

The second path that is proposed is a sinusoidal path in the Cartesian space. It is defined as

$$p_{ref}(s) = \begin{pmatrix} 0.3 + 0.2 \cdot s \\ -0.1 + \frac{s}{3} + 0.2 \cos(4\pi \cdot s) \\ 0.2 + 0.25 \cdot s \end{pmatrix} \tag{5.20}$$

Again, the desired Cartesian velocity and acceleration can be written by time differentiation of Equation (5.20). It results

$$\dot{p}_{ref} = \begin{pmatrix} 0.2 \cdot \dot{s} \\ \frac{\dot{s}}{3} - 0.2 \sin(4\pi \cdot s) \cdot \dot{s} \\ 0.25 \cdot \dot{s} \end{pmatrix} \tag{5.21}$$

$$\ddot{p}_{ff} = \begin{pmatrix} 0.2 \cdot \ddot{s} \\ \frac{\ddot{s}}{3} - 0.2(4\pi)^2 \cos(4\pi \cdot s) \cdot \dot{s}^2 - 0.2(4\pi) \sin(4\pi \cdot s) \cdot \ddot{s} \\ 0.25 \cdot \ddot{s} \end{pmatrix} \tag{5.22}$$

Again, a the orthographic projection of this task can be seen in Figure 5.3.

5.2.4 Cartesian Heart-Shaped Path

Following, the third and last path is represented by an heart drew in the Cartesian space. It is defined as a quite complex trajectory, combining several sines and cosines as follows

$$p_{ref}(s) = \text{center} + \frac{1}{4} \begin{pmatrix} \sin(2\pi \cdot s)^3 \\ \frac{1}{16}(13 \cos(2\pi \cdot s) - 5 \cos(4\pi \cdot s) - 2 \cos(8\pi \cdot s)) \\ 0 \end{pmatrix} \tag{5.23}$$

around a center

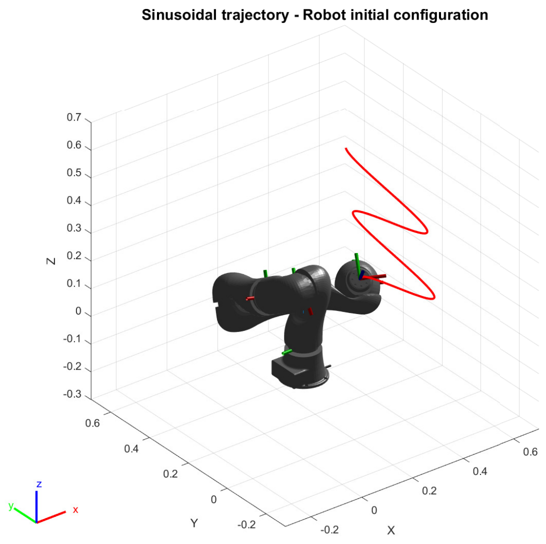
$$\text{center} = \begin{pmatrix} 0.55 \\ 0.1 \\ 0.3 \end{pmatrix} \tag{5.24}$$

Differentiating with respect to time, the desired Cartesian velocity and acceleration can be written as

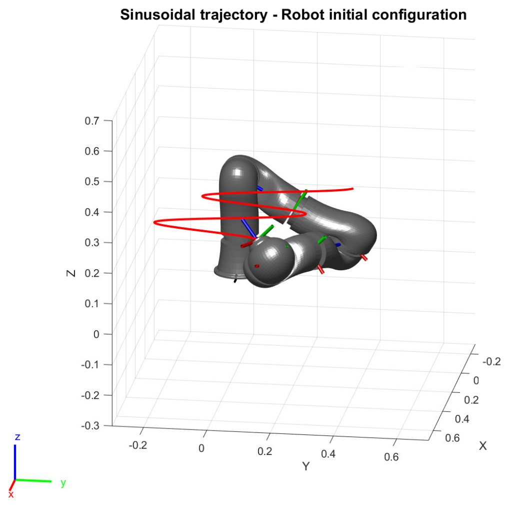
$$\dot{p}_{ref} = \begin{pmatrix} \frac{1}{2}(3\dot{s}\pi \cos(2\pi \cdot s) \sin(2\pi \cdot s)^2) \\ \frac{1}{8}\dot{s}(\frac{(5\pi \sin(4\pi \cdot s))}{2} - \frac{(13\pi \sin(2\pi \cdot s))}{4} + \frac{(3\pi \sin(6\pi \cdot s))}{2} + \pi \sin(8\pi \cdot s)) \\ 0 \end{pmatrix} \tag{5.25}$$

$$\ddot{p}_{ff} = \begin{pmatrix} \frac{3}{2}\pi\dot{s}\ddot{s} \cos(2\pi \cdot s) \sin(2\pi \cdot s)^2 - (3\dot{s}\pi^2 \sin(2\pi \cdot s)^3 - 6\dot{s}\pi^2 \sin(2\pi \cdot s) \cos(2\pi \cdot s)^2)\dot{s} \\ \frac{(5\pi^2 \cos(4\pi \cdot s))}{4} - \frac{13\pi^2 \cos(2\pi \cdot s)}{16} + \frac{9\pi^2 \cos(6\pi \cdot s)}{8} + \pi^2 \cos(8\pi \cdot s)\dot{s}^2 + \\ (\frac{(5\pi \sin(4\pi \cdot s))}{16} - \frac{(13\pi \sin(2\pi \cdot s))}{32} + \frac{(3\pi \sin(6\pi \cdot s))}{16} + \frac{\pi \sin(8\pi \cdot s)}{8})\dot{s}\ddot{s} \\ 0 \end{pmatrix} \tag{5.26}$$

Finally, a graphical representation is shown in Figure 5.4.

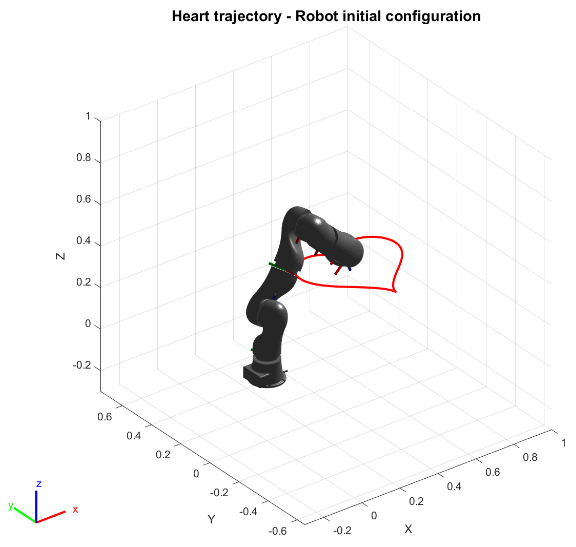


(a) Orthographic projection

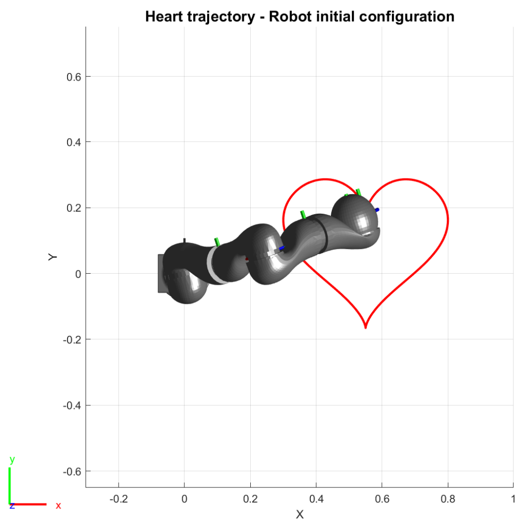


(b) Side view

Figure 5.3. Graphical representation for the Cartesian sinusoidal task.



(a) Orthographic projection



(b) X-Y view

Figure 5.4. Graphical representation for the Cartesian heart-shaped task.

5.3 Results for the Cartesian Periodic Circular Trajectory

First, the results about the simulation for the Cartesian circular task are reported. It is important to notice that, due to the redundancy, even if the task is periodic, in general the repeatability for the joint motion does not hold. As a consequence, after one tour, the robot will find itself in a different state when passing through the same Cartesian point, affecting the variance. However, in the following the main results are presented.

First, in Figure 5.5 the motion of the robot under the pure model based control law and the improved one with the learning correction are compared with the desired motion by the use of three-dimensional scatter plots.

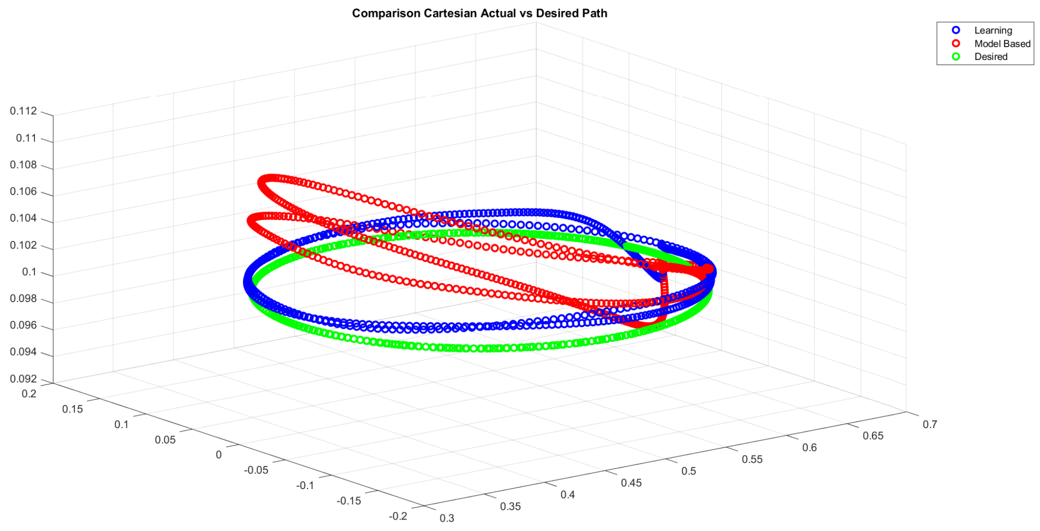


Figure 5.5. Comparison of the robot motion with and without learning correction.

In particular, the green curve represents the desired end-effector motion, while the red and the blue ones represent the actual robot's end-effector motion under the pure model based control and the one augmented with the learning prediction. It is clear how the error using the one with the correction term provided by the Gaussian processes results to be much lower with respect to the error provided by applying a model based technique.
In order to better quantify the difference in terms of the end-effector Cartesian error, Figure 5.6 compares axis by axis the Cartesian errors.
Again, the improvement obtained by using the learning correction appears as evident if compared with the nominal control law.

It is interesting to investigate also the effects of the variance optimization, both on the variance itself and on the accuracy of the prediction mean.

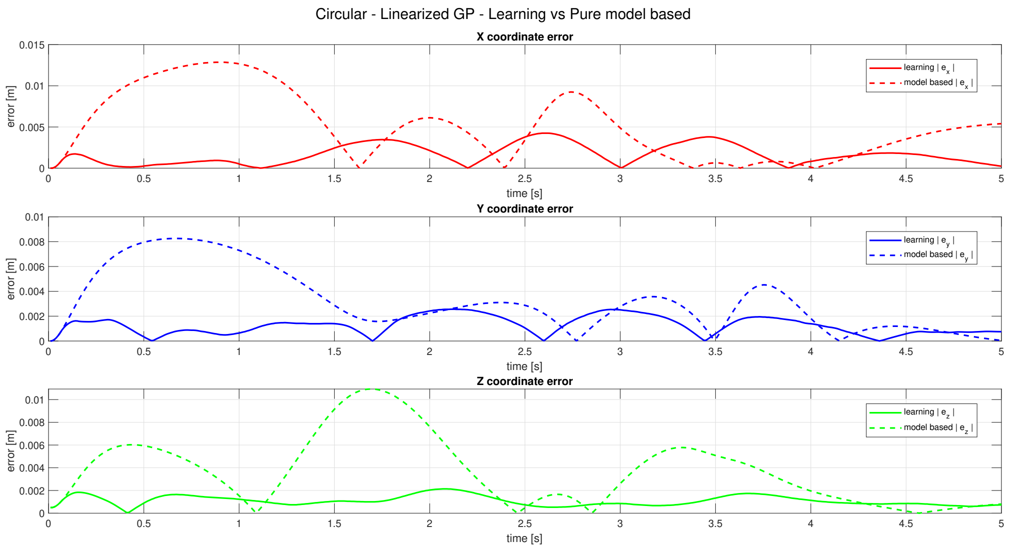
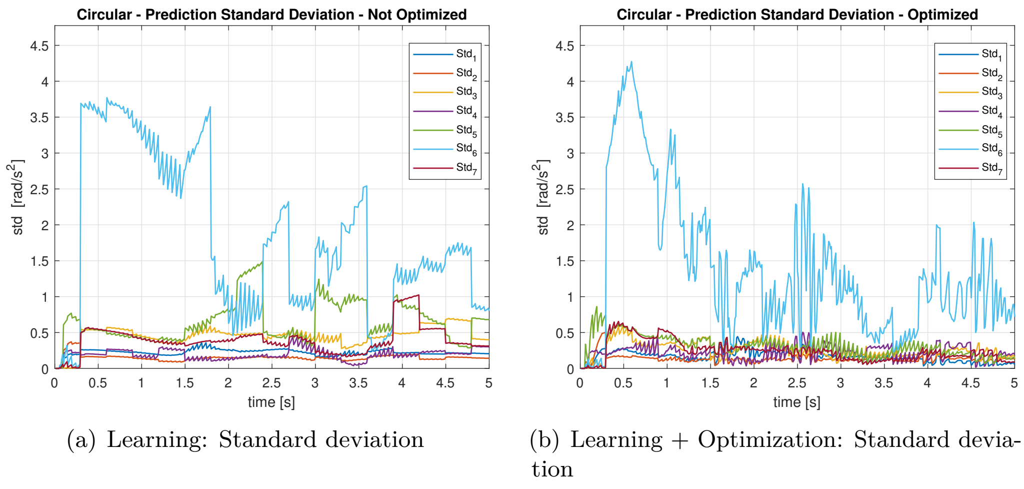


Figure 5.6. Cartesian components of the end-effector tracking error.

Indeed, as shown in Figure 5.7, the prediction standard deviation gets successfully lowered for each Gaussian process by the variance optimization in the robot's Jacobian null-space.



(a) Learning: Standard deviation

(b) Learning + Optimization: Standard deviation

Figure 5.7. Standard deviation comparison with and without variance optimization.

Moreover, to have a more reliable comparison between the standard deviation with and without the variance optimization, in Figure 5.8 the quadratic norm of the prediction standard deviations, i.e. the sum of the prediction variances, provided by the two approaches has been compared.
Hence, the overall prediction standard deviation has been minimized, but, one may wonder how this variance improvement can improve the quality of the predictions.

As a recall, the variance can be considered as a measure of the prediction uncertainty, implying a low-variance region to be more reliable with respect to

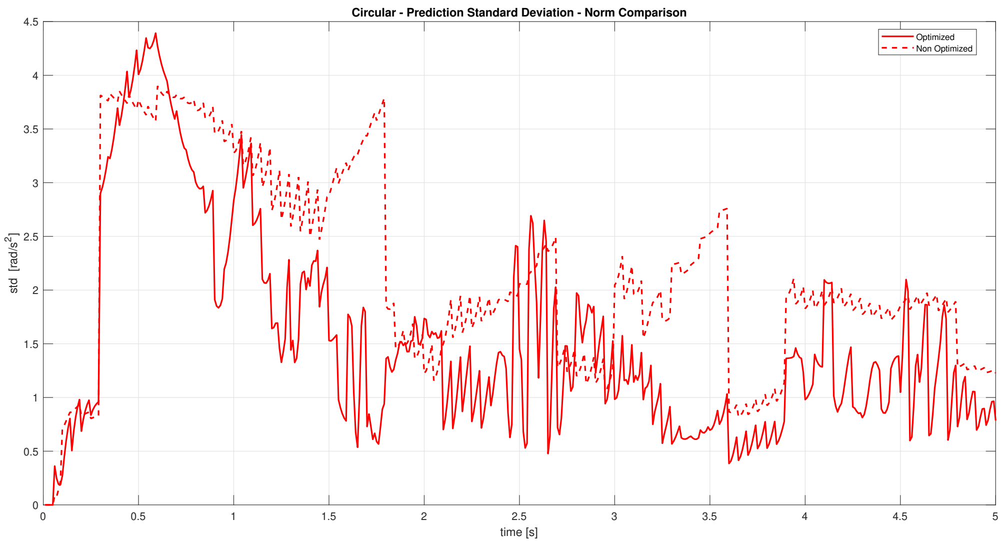
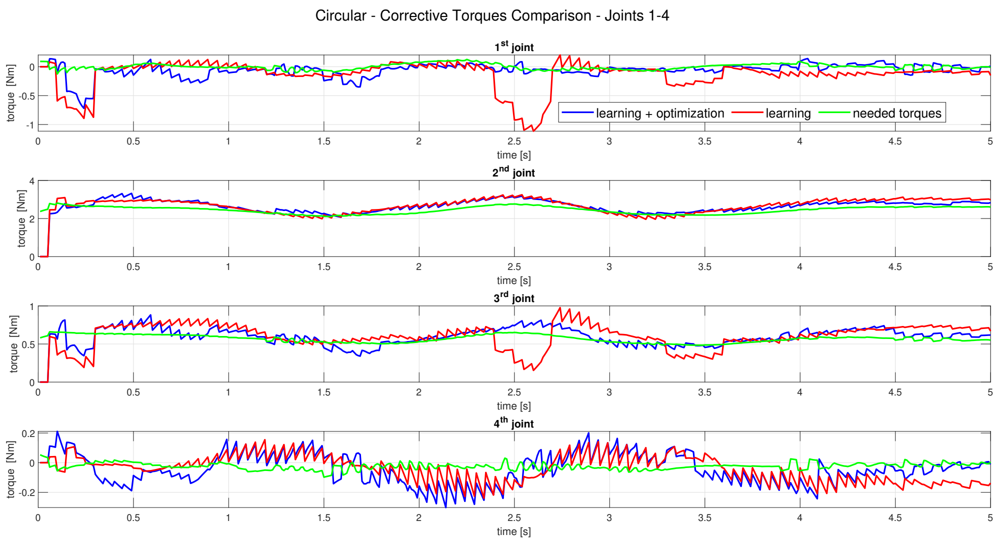


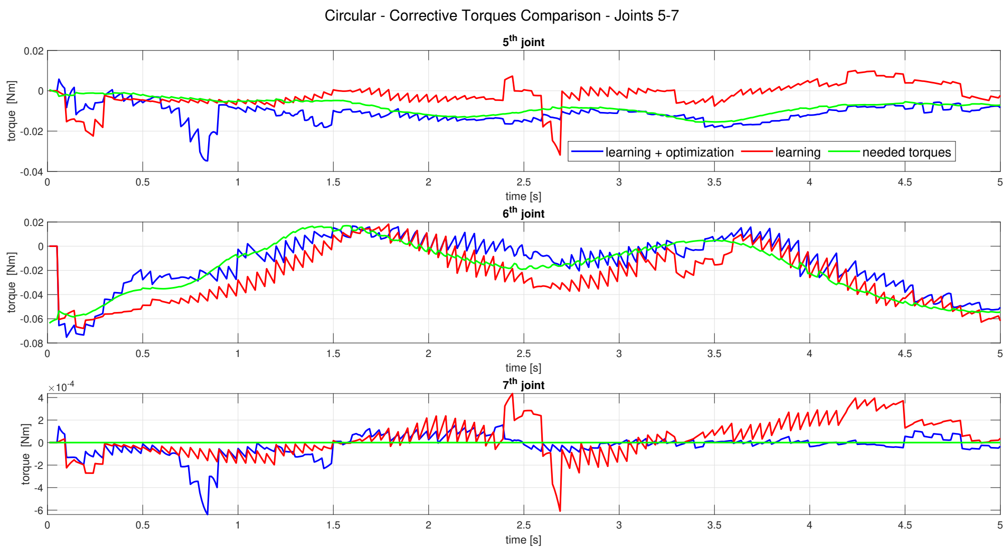
Figure 5.8. Comparison between the quadratic norm of the prediction standard deviations with and without null-space variance optimization.

another one with higher variance. In particular, this can be easily checked looking at the comparison between the torque predictions provided by the Gaussian processes with and without this kind of optimization with respect to the real torques missing because of the model uncertainty. In particular, this comparison is shown joint-by-joint in Figure 5.9, where the green line represents the needed torques and the blue and the red ones represent the corrective torques predicted with and without the null-space variance optimization respectively.

Finally, it is worth to notice that the Gaussian processes predictions are in terms of joint accelerations, but they can be easily converted to the corresponding torques by using the dynamic model. Also, the real needed torques can be easily computed in a simulation context for analysis purposes by using the information about $\Delta M(q)$ and $\Delta\eta(q,\dot{q})$.



(a) Joints 1-4



(b) Joints 5-7

Figure 5.9. Comparison between the predicted corrective torque with and without null-space variance optimization with respect to the real needed ones.

5.4 Results for the Cartesian Sinusoidal Trajectory

The next proposed task is represented by the Cartesian sinusoidal trajectory, again a three-dimensional task (only the position matters) guaranteeing a four-dimensional Jacobian null-space. As for the previous task, the main results are reported below. First of all, the Figure 5.10 shows the actual robot motion with and without learning compared with the desired one, where the blue and the red lines represent the actual end-effector Cartesian motion with and without the learning correction, and the green line represents the desired end-effector path.

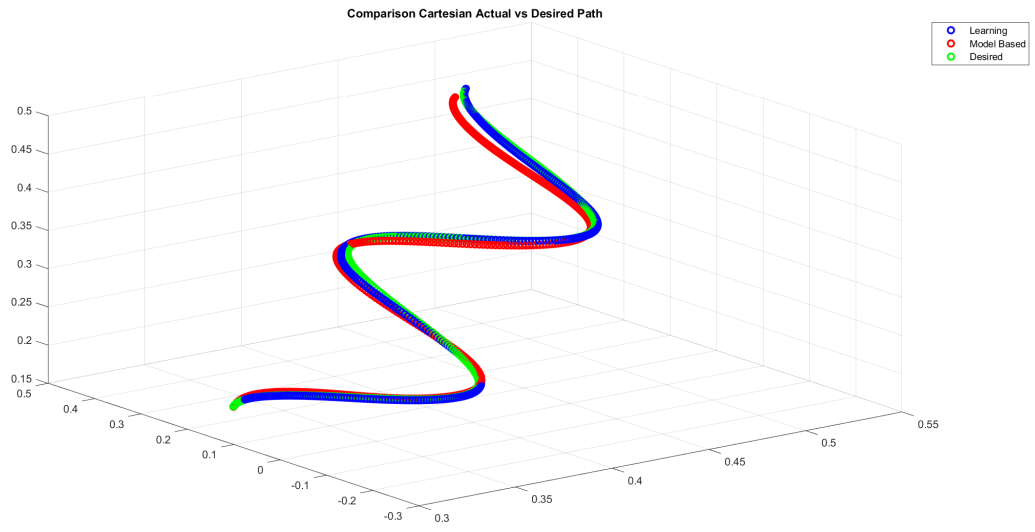


Figure 5.10. Comparison of the robot motion with and without learning correction.

Again, the error using the the control law merging the model-based and the learning approaches results to be lower with respect to the one provided by simply the model-based control on the nominal model.
To quantify the error for both the approaches, in Figure 5.11 the Cartesian errors are shown axis by axis. Indeed, a large improvement can be noticed, in particular for the x and the z axes, where the errors approach zero.

Looking at the improvement brought by the variance optimization, instead, in Figures 5.12 and 5.13, a comparison between the standard deviation components and their quadratic norm are reported, highlighting a significant decrease of the overall prediction standard deviation.

Finally, Figure 5.14 reports joint-by-joint the comparison between the predicted torque corrections provided by the linearized Gaussian processes with and without the null-space optimization term with respect to the ones which are needed for fully compensating the dynamic errors introduced by the model uncertainties. In particular, the green, blue and red lines represent the needed torques and the corrective torques predicted with and without the null-space variance optimization respectively, showing a strong improvement also on the accuracy of the prediction means.

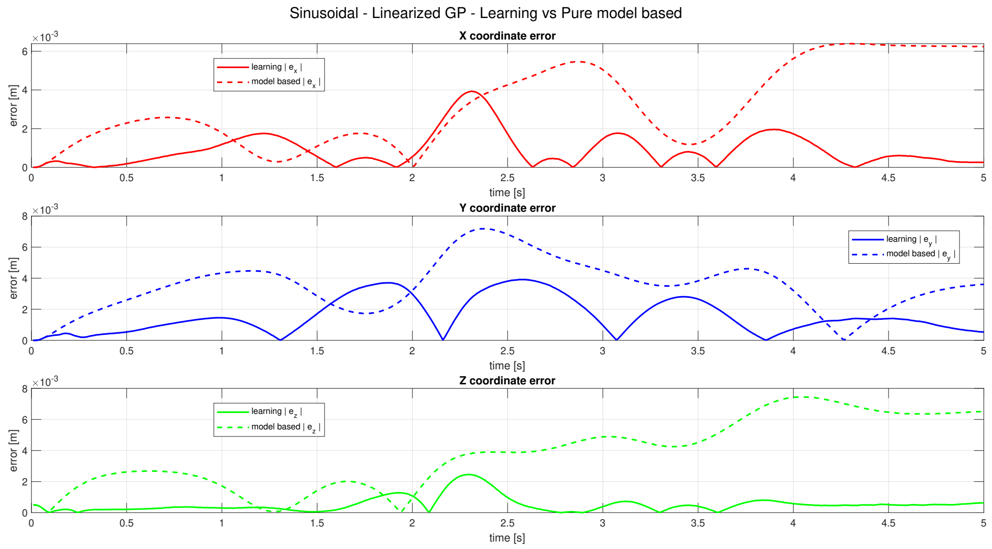
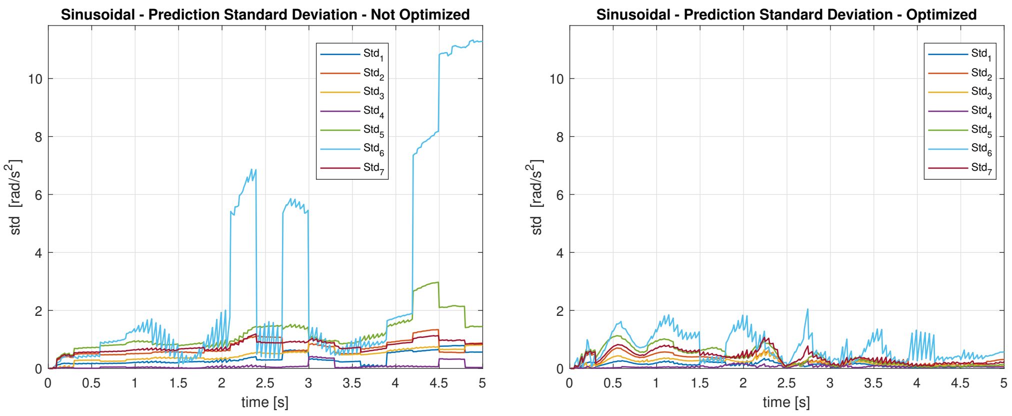


Figure 5.11. Cartesian components of the end-effector tracking error.



(a) Learning: Standard deviation (b) Learning + Optimization: Standard deviation

Figure 5.12. Standard deviation comparison with and without variance optimization.

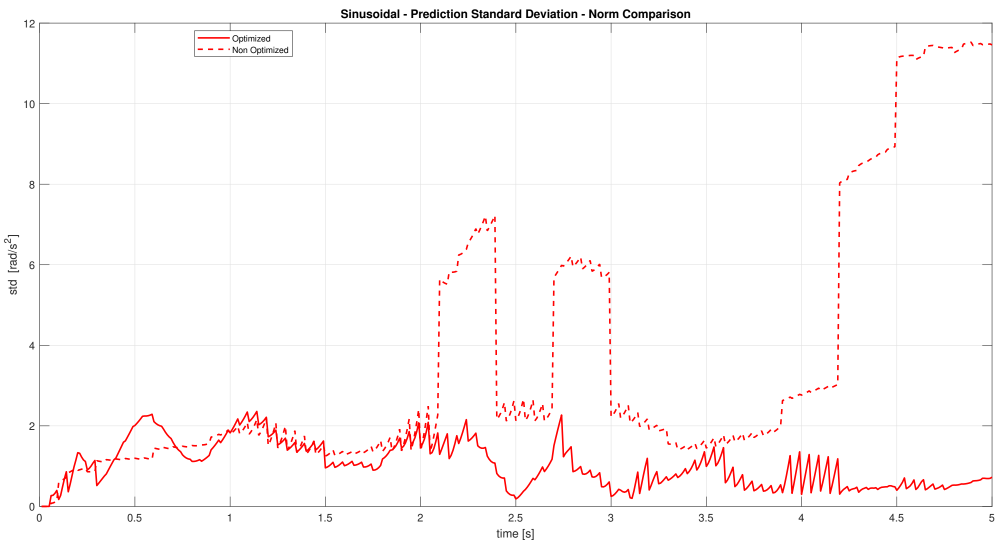
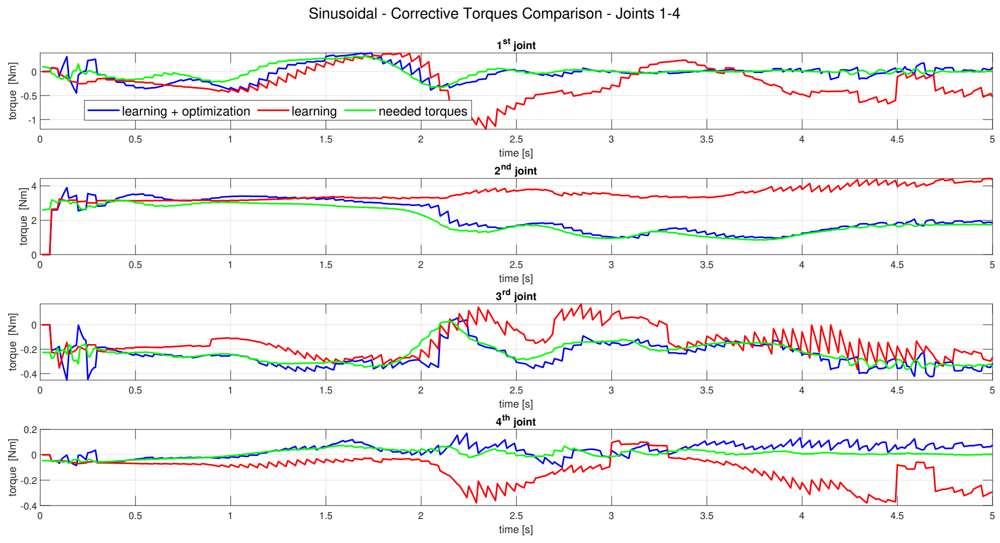
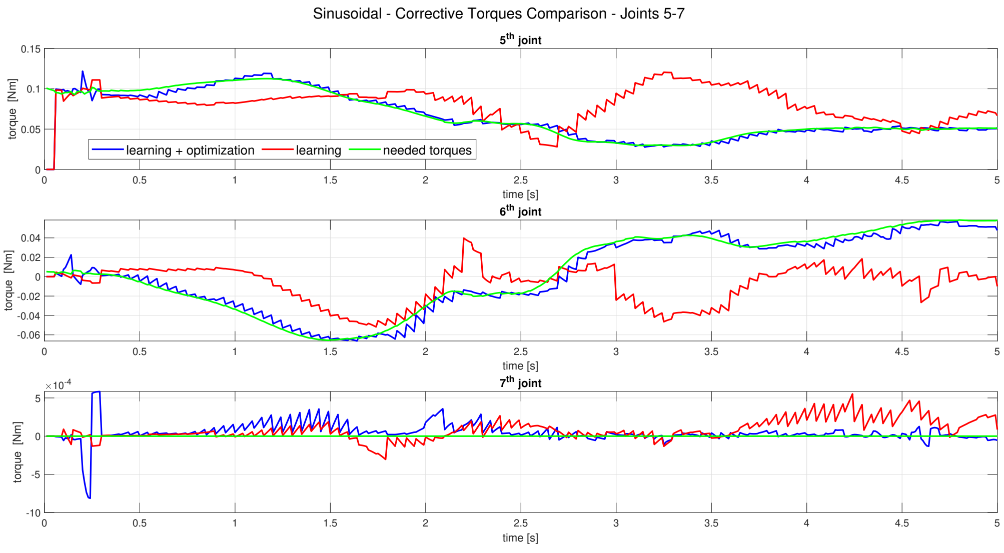


Figure 5.13. Comparison between the quadratic norm of the prediction standard deviations with and without null-space variance optimization.



(a) Joints 1-4



(b) Joints 5-7

Figure 5.14. Comparison between the predicted corrective torque with and without null-space variance optimization with respect to the real needed ones.

5.5 Results for the Cartesian Heart-Shaped Trajectory

Lastly, a more articulated task is proposed. It is still a three-dimensional task in the Cartesian space, but having the shape of an heart. From a kinematic point of view, this implies also the presence of discontinuities, i.e. an inversion in the velocity during the motion. As for the previous tasks, the main results are here reported.

In first place, in Figure 5.15 the actual robot motion under the pure model-based and the improved one are compared with the desired motion. In particular, the blue and the red lines represent the actual end-effector Cartesian motion with and without the learning correction, and the green line represents the desired one.

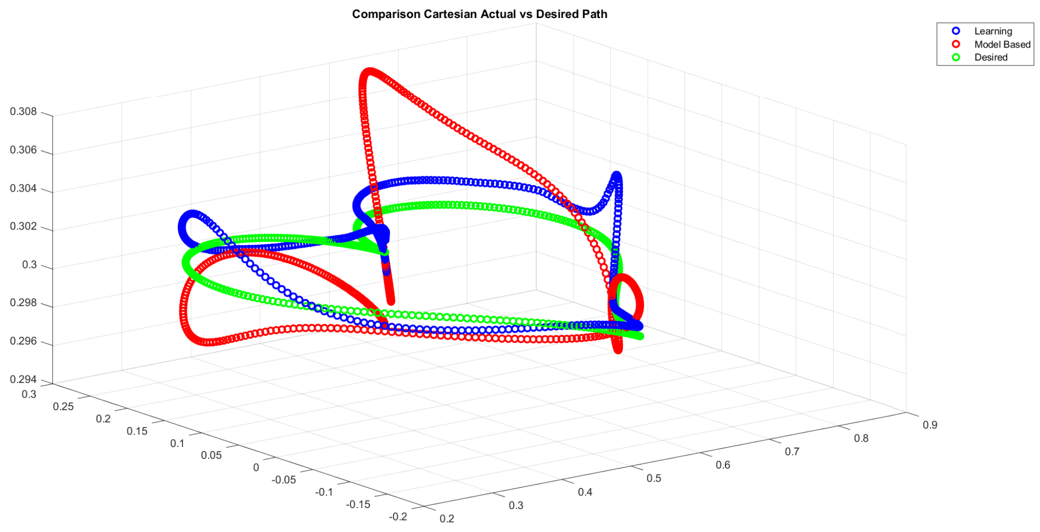


Figure 5.15. Comparison of the robot motion with and without learning correction.

As before, augmenting the nominal control law with the learning correction, the error gets lowered with respect to the one provided by just the nominal control law.

In order to give a measure of the error for both the approaches, Figure 5.16 shows Cartesian components of the end-effector tracking error. Here, an improvement can be noticed on each axis, but there is still a peak between 1.5 and 2 seconds. This is because the robot is approaching a kinematic singularity. Indeed, Figure 5.17 shows the time evolution of the Jacobian smallest singular value, and in those particular instants it rapidly falls. However, thanks to the use of the damped least square solution, by paying as cost the introduction of a little error, the reference and actual velocities does not increase too much.

About the improvement brought by the variance optimization, Figures 5.18 and 5.19 compare the standard deviation components and their quadratic norm obtained without and with the null-space optimization respectively, revealing a slight decrease of the overall uncertainty, still affected by the vicinity of the kinematic singularity in the period between 1.5 and 2 seconds.

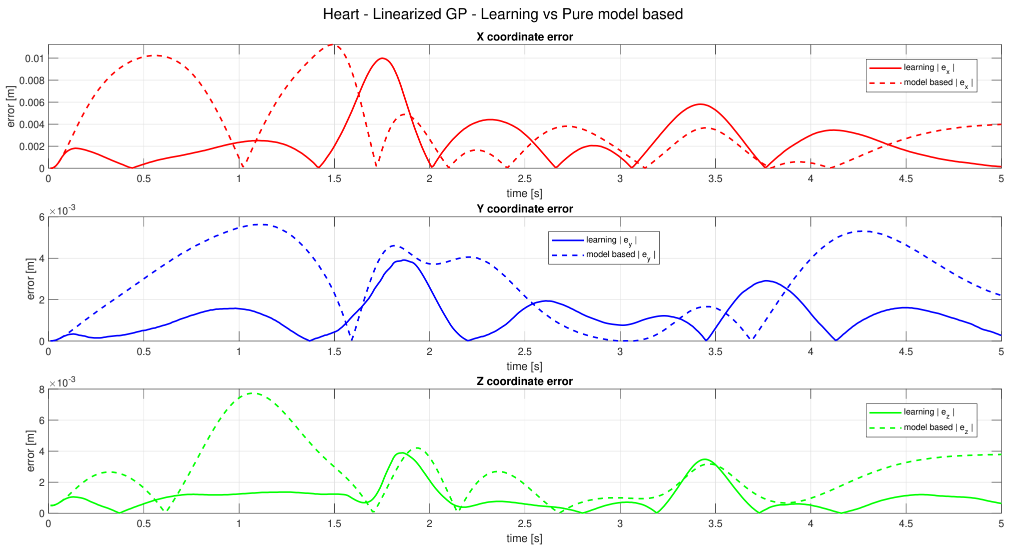


Figure 5.16. Cartesian components of the end-effector tracking error.

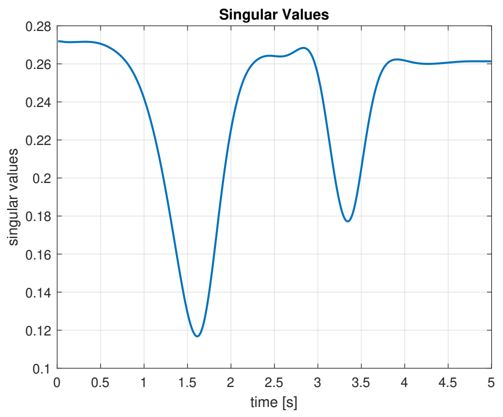
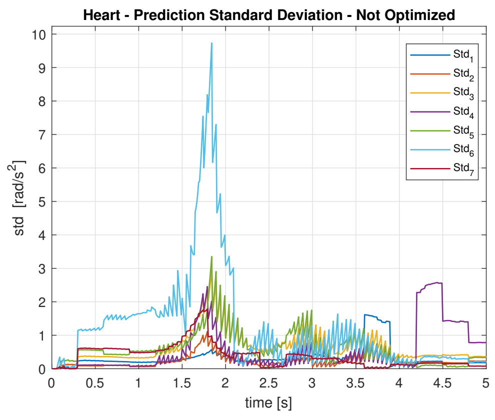
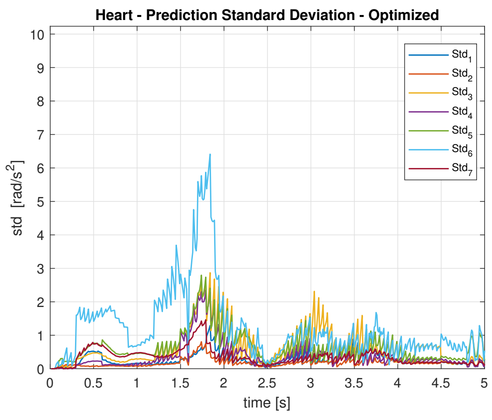


Figure 5.17. Time evolution of the Jacobian smallest singular value.

Finally, in Figure 5.20 there are shown and compared joint-by-joint both the predicted torque corrections with and without the null-space variance optimization and the torques which are needed to fully compensate the error due to the model uncertainties. In particular, the green, blue and red lines represent the needed torques and the corrective torques predicted with and without the null-space variance optimization respectively, reflecting the previous improvement also on the prediction accuracy.



(a) Learning: Standard deviation



(b) Learning + Optimization: Standard deviation

Figure 5.18. Standard deviation comparison with and without variance optimization.

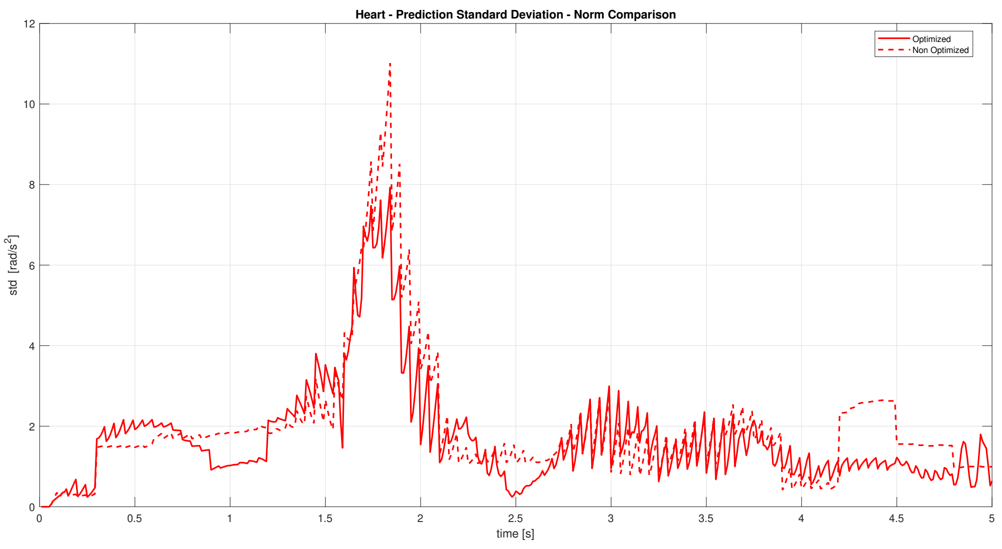
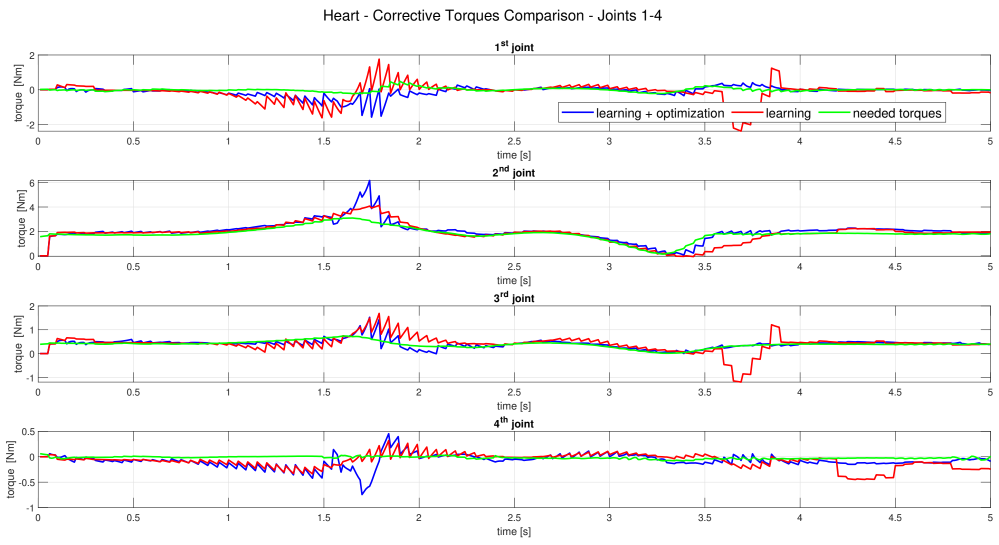
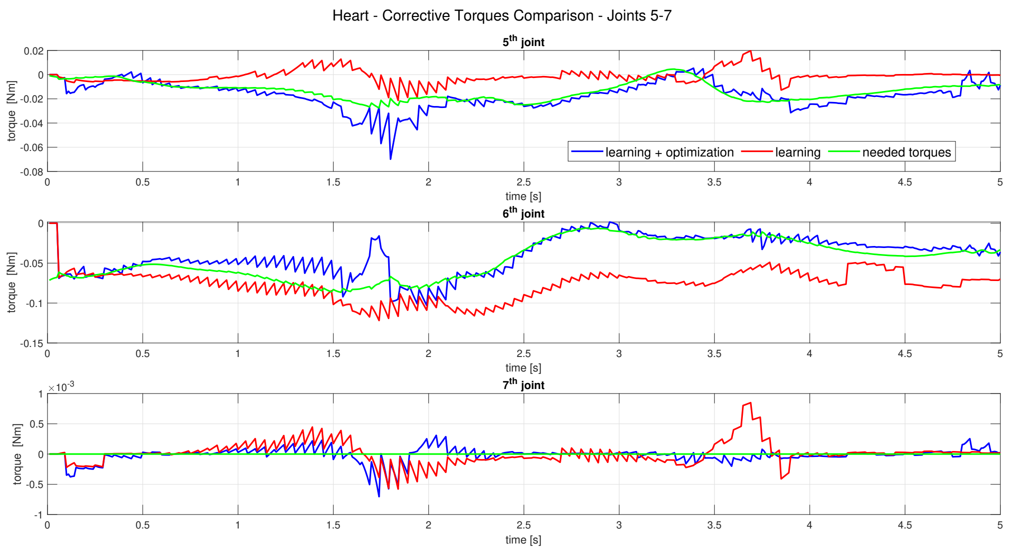


Figure 5.19. Comparison between the quadratic norm of the prediction standard deviations with and without null-space variance optimization.



(a) Joints 1-4



(b) Joints 5-7

Figure 5.20. Comparison between the predicted corrective torque with and without null-space variance optimization with respect to the real needed ones.

Chapter 6

Friction Estimation

In the previous chapter, a large uncertainty on the masses was considered, but friction was ignored both on the nominal and on the real models. Consider now a more realistic case, where both the unmodeled frictions and the masses uncertainties are present.
In particular, this chapter is organized as follows. First, a recap is proposed about the operational settings used for this experiment, defining both the mass uncertainties and how the friction affects the real model; following, the results of the simulation are reported with some observations. Finally, a comparison between the learning performance and the nominal one with the introduction of an additive kinematic reference component in the null space is reported.

6.1 Simulation Setup

Consider the Cartesian sinusoidal task introduced in the previous chapter.
First of all, the dynamic model of the real robot including friction can be written as

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

where $\tau_f(\dot{q})$ represents the friction torque affecting the model.
In a compact way, the previous formula can be rewritten as

$$M(q)\ddot{q} + \eta(q,\dot{q}) = \tau \tag{6.2}$$

where η takes into account also the friction torque.
Being in a simulation context, there is a need to model the friction torques.
In this work, they have been modeled as the sum of a shifted sigmoid and a linear function of joint velocities. Mathematically speaking, they have been represented as

$$\tau_f = \frac{1}{2}\alpha_f \frac{1}{1+\mathrm{e}^{-30\dot{q}}} + \beta_f\dot{q} - \frac{1}{2}\mathbf{1}_{N\times 1} \tag{6.3}$$

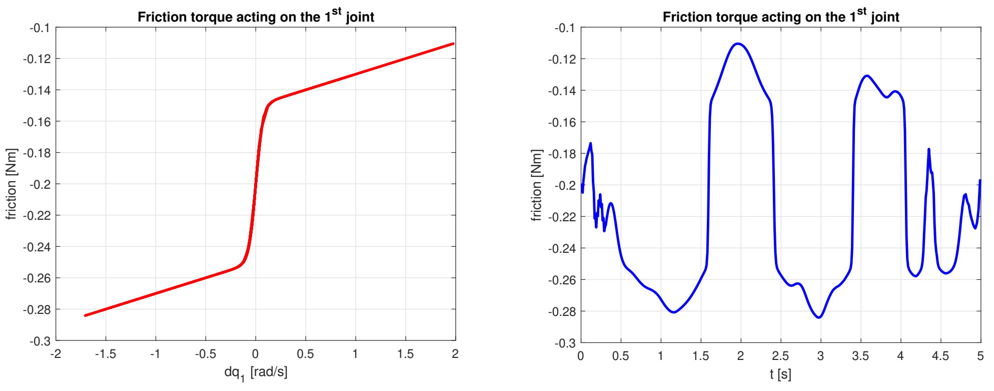
where $\mathbf{1}_{N\times 1}$ represents an N-by-1 unitary vector and α_f and β_f are diagonal matrices defined as

$$\alpha_f = \begin{pmatrix} 0.2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.08 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.08 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.001 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.0001 \end{pmatrix} \tag{6.4}$$

and

$$\beta_f = 0.2\ \alpha_f \tag{6.5}$$

respectively.
As an example, the behavior with respect to the joint velocities and to time of the so modeled friction torque acting on the first joint is shown in Figures 6.1(a) and 6.1(b) respectively. Of course, for all the other joints, it does hold a similar behavior.



(a) Joint velocity profile of the friction torque (b) Time profile of the friction torque

Figure 6.1. Friction torque profile as a function of joint velocity and its time evolution on the first joint.

Of course, the information about the friction torques will be used only for simulating the real dynamics, while for the nominal model used for controlling the robot it will be assumed to have completely unmodeled frictions.
As a consequence, the term $\Delta\eta$, defined as

$$\Delta\eta(q, \dot{q}) = \eta(q, \dot{q}) - \hat{\eta}(q, \dot{q}) \tag{6.6}$$

can be rewritten as

$$\Delta\eta(q, \dot{q}) = \Delta c(q, \dot{q}) + \Delta g(q) + \tau_f(\dot{q}) \tag{6.7}$$

that, under the assumption of having a perfect dynamic model, i.e. without masses uncertainties, implying also

$$\Delta M = \Delta\eta = 0 \tag{6.8}$$

corresponds exactly to the friction torque τ_f. Hence, the goal of this experiment will be to successfully predict $\Delta\eta$.

It is worth to remark also that, for analysis purposes, being in a simulation context, $\Delta\eta$ can be easily computed.

About the masses uncertainties, instead, they are listed as follows:

- +30% on the first mass;
- +30% on the second link mass;
- -30% on the third link mass;
- -30% on the fourth link mass;
- +30% on the fifth link mass;
- -30% on the sixth link mass;
- +30% on the seventh link mass.

Moreover, differently with respect to the previous simulations, a pseudo-inverse solution for solving redundancy has been used instead of the DLS one. It other words, it is equivalent to set the damping term μ^2 of the DLS solution to zero. Finally, the linearization step has been set from 5 to 2, in order to have more accurate linearized Gaussian processes predictions.

6.2 Simulation Results

In this section the results of the simulation are reported.

First, the end-effector motion under the two control laws is compared in Figure 6.2, where the green line represents the desired motion, and the blue and the red lines the motion obtained with and without learning respectively. It is evident how the presence of friction, if only the nominal control law is applied, significantly deteriorates the performance, and large errors appear. This is more than attenuated by the learning correction, that greatly improves the tracking of the desired trajectory.

Going on, to quantify this error on each axis, the Cartesian tracking mismatch between the desired and the actual position has been shown in Figure 6.3, highlighting the great improvement brought by the proposed learning approach.

Following, it is interesting to notice how the linearized Gaussian processes successfully predict the $\Delta\eta$ quantity, that, as a recall, contains information about the friction torque. In particular, being in a simulation context, it is easy to isolate the $\Delta\eta$ term from the full expression representing the torques that are needed to compensate the whole uncertain dynamics, obtaining the results represented in Figure 6.4, showing the effectiveness of this approach.

Finally, it is worth to remark that, if a perfect dynamic model is available, the $\Delta\eta$ term will become exactly the friction torque, allowing for a fast and an accurate estimate of the friction itself.

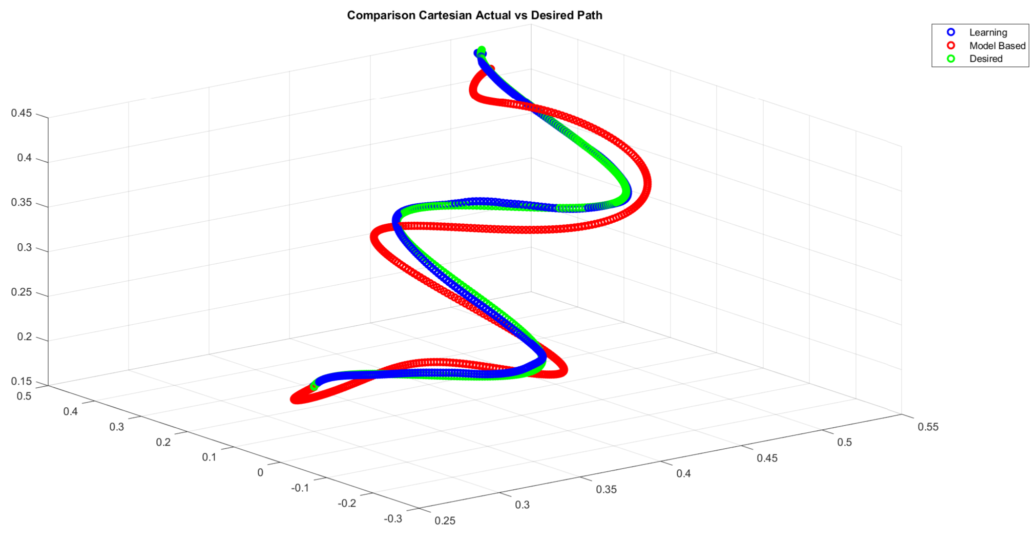


Figure 6.2. Robot motion with and without the learning correction.

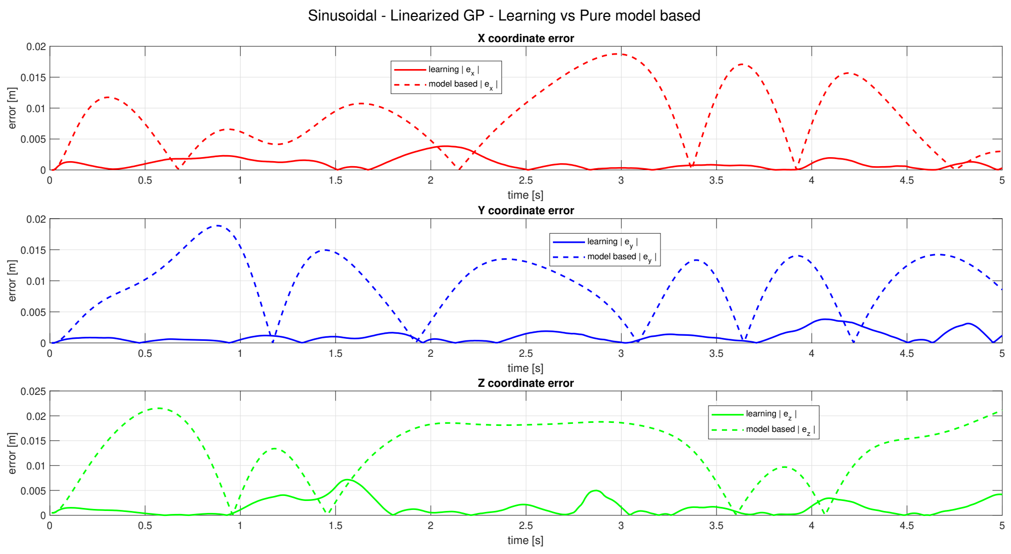
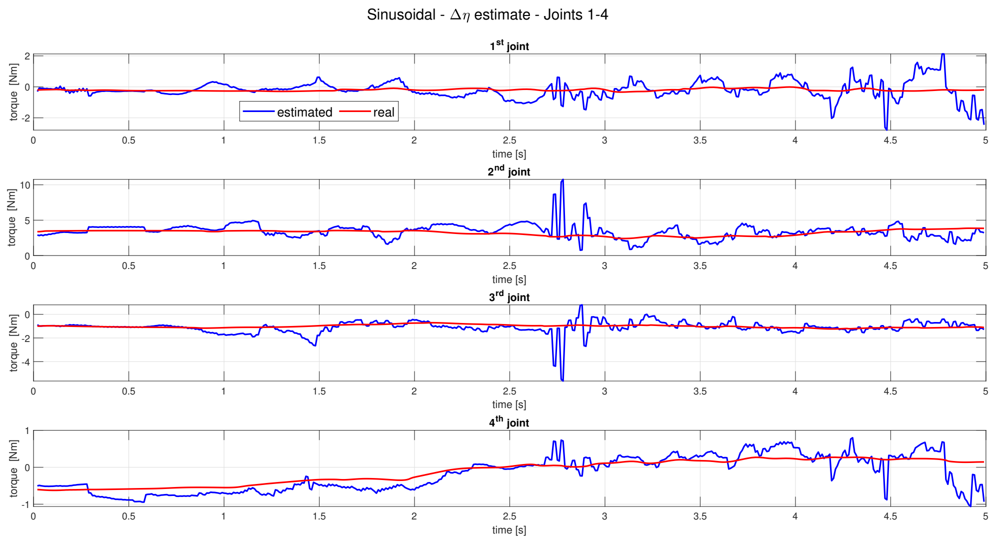
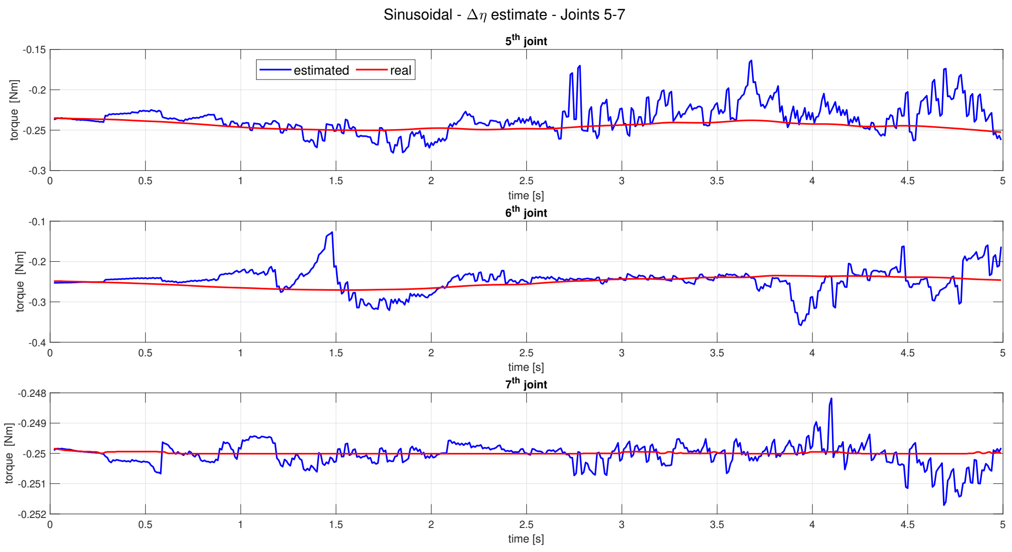


Figure 6.3. Cartesian components of the end-effector tracking error.



(a) $\Delta\eta$ estimate for joints from 1 to 7



(b) $\Delta\eta$ estimate for joints from 5 to 7

Figure 6.4. Prediction of the difference $\Delta\eta$ by the use of the linearized Gaussian processes.

6.3 Introduction of a Null-Space Kinematic Reference

On the previous simulations, the advantages provided by the learning correction were highlighted. To push this result even more, in this section, the nominal control, without the learning correction, is improved by adding a stabilizing kinematic reference in the null-space of the robot Jacobian. As a recall, a damping term acting on the velocity was already present with the aim of stabilizing the self-motions.

The idea is to add in the null-space a reference for the joint configurations. In particular, this stabilizing term takes the form as follows

$$K_P(q_{ref} - q), \qquad K_P > 0 \tag{6.9}$$

where the reference q_{ref} has been chosen as the joint configurations obtained in the previous simulation including also the learning correction, and the gain K_P has been set to

$$K_P = 100 \tag{6.10}$$

Hence, the robot will be pushed so as to let it assume the same joint configurations obtained in the simulation where the learning correction was taken into account.

In Figure 6.5 the end-effector motions, obtained with and without the learning correction, are compared with the desired motion. In particular, the green line represents the desired performance, the blue line represents the motion obtained by including also the learning correction and the red one the motion obtained under the nominal controller with the kinematic reference expressed as in Equation (6.9) projected in the null-space. As a result, it is clear how the introduction of the kinematic reference improved a bit the tracking performance, also stabilizing the joint motions, but there is still a significant tracking error due to the uncertain dynamics that the nominal controller is not able to compensate, no matter how many kinematic terms are introduced in the robot Jacobian null-space.

In order to have a measure of the tracking Cartesian error and to quantitatively compare the performance obtained with and without the learning correction, instead, Figure 6.6 shows the tracking error for each Cartesian component. The solid and the dashed line represent respectively the learning performance and the nominal one, checking again a significant reduction of the end-effector tracking error provided by taking into account also of the learning correction.

So, once again, the learning correction resulted to achieve the best performance, almost matching the desired one.

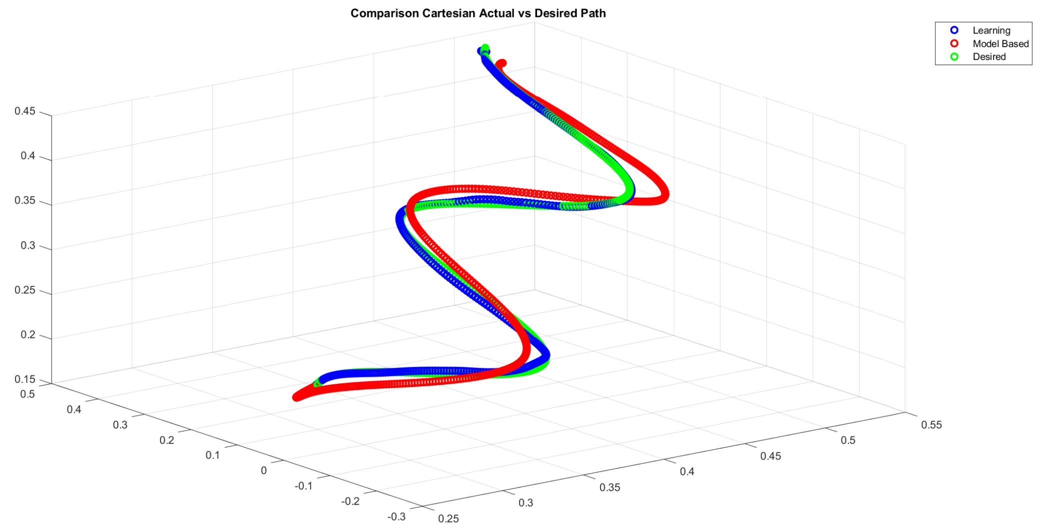


Figure 6.5. Robot motion with and without the learning correction.

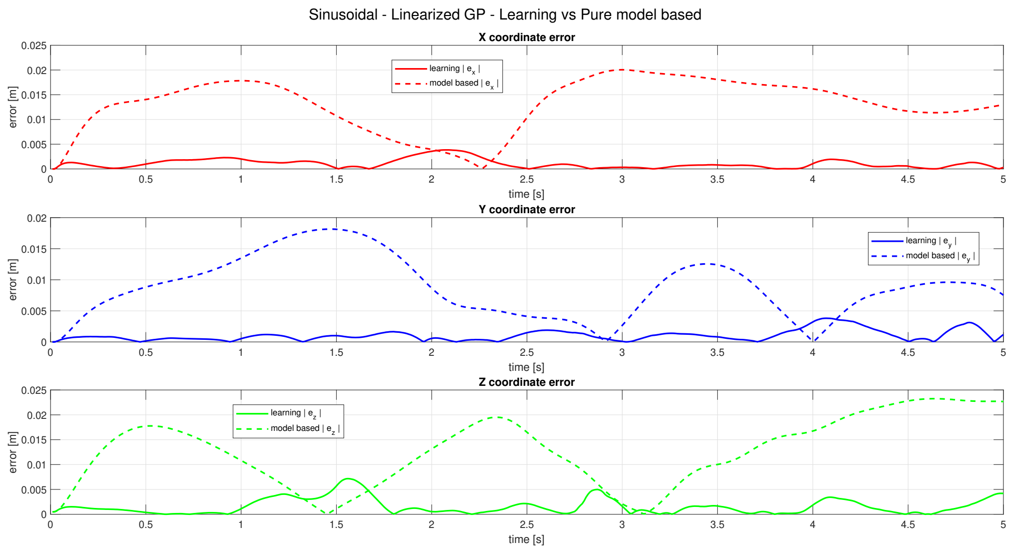


Figure 6.6. Cartesian components of the end-effector tracking error.

Chapter 7

Conclusions

In this chapter, the main topics, together with the problem description and how it has been handled are briefly recalled.

First of all, a relatively in-depth overview about the robot kinematics and dynamics was given, defining a mapping, both direct and inverse whether possible, between the joint and the task space and relating the generalized torques to the actual robot motion through the dynamic model. In particular, a great deal of attention has been paid to what the kinematic redundancy is and how to handle it in order to achieve some desired objectives, playing a key role in the algorithm later discussed. Then, the robot control has been introduced, focusing mainly on the 'Computed Torque' control law and the feedback linearization technique, exploring the critical issues faced in presence of dynamic uncertainty. Indeed, if there is a mismatch between the real and the nominal dynamic parameters, or if there are some unmodeled frictions, there will be still some nonlinear terms that will not be cancelled by the nominal feedback linearization, deteriorating the performance scored by the nominal model-based controller.
This overview was then mandatory in order to provide the theoretical background needed to fully understand the problem this thesis deals with.

Following, the problem has been formulated as to find an estimate of the regressor representing all the unmodeled dynamic terms, in order to estimate the dynamic uncertainties and to compensate for them to allow a perfect feedback linearization. To this purpose, a machine learning approach has been proposed. In the literature there are several methods that can suit this kind of problem, such as linear regression, regression Support Vector Machine, Neural Networks, but in this thesis the selected machine learning approach was the Gaussian Process regression. Indeed, the choice of a regressor based on Gaussian processes was strongly linked to the property of a Gaussian process to provide accurate prediction estimates even with small datasets, allowing for an on-line approach, without a strictly needed training phase. Also, the Gaussian processes provide information about the prediction variance, representing a measure of the estimation error. Hence, recalling the robot to be redundant with respect to the desired task, this redundancy can be handled and exploited in order to minimize the variance, allowing for more accurate predictions.

Also, because of the cubic scaling of the Gaussian process computational time with respect to the dimension of the dataset, in order to significantly reduce this cost and making the algorithm more reliable at the real-time level, a sparse linearized Gaussian process has been proposed and implemented. This allowed for writing the variance as a quadratic form, leading the way to the definition of an efficiently solvable Linear Quadratic optimization problem in terms of the joint acceleration. In particular, the optimization problem was defined taking into account and suitably weighting both the variances at the current time and at the next step. As a result, to find the solution of this LQ problem provided a huge speed-up with respect to its nonlinear version, significantly increasing also the prediction accuracy.

Subsequently, the results of the simulations with respect to the desired tasks and assuming different dynamic uncertainties have been reported.
In particular, in first place it was assumed to have a large uncertainty on the link masses, up to 60%, and the performance improvement brought by augmenting the nominal model-based approach with the learning correction was validated through the simulation of the robot executing different tasks in the Cartesian space, i.e. a periodic circular trajectory, a sinusoidal and an heart-shaped one. As a result, the missing and needed torque due to the uncertainty was successfully predicted by the Gaussian processes, and the tracking error was significantly reduced on each axis.
Following, a more realistic case was considered. Indeed, it was assumed to have a 30% of uncertainty on each mass and the presence of unmodeled frictions. In particular, the friction on the real model was modeled as the sum of a shifted sigmoid plus a linear term with respect to the joint velocities, while in the nominal model, it was considered as completely unknown.
The simulation was performed on the Cartesian sinusoidal task, on which this kind of uncertainty resulted into a severe worsening of the tracking performance under a nominal model-based approach, without the learning correction. On the other hand, the learning approach really shone, highlighting and confirming a huge improvement in reducing the Cartesian tracking error, from the order of several centimeters, to just few millimeters. Moreover, under the assumption of having a perfect dynamic model, without uncertainties on the dynamic parameters, the Gaussian processes prediction results into a reliable estimate of the friction itself.

Finally, it is important to point out one of the advantages brought by this learning approach that is also related to safety. Indeed, under the assumption of a self-collision-free planning, the nominal model-based approach, as opposed to the one improved with the learning correction, can lead the robot to self-collide, that in a real case would fail the execution of the proposed task. Also, in the previous simulations, self-collisions were ignored in order to show up the tracking improvements, but if the same experiments are run while checking and considering self-collisions, the robot controlled without the learning correction will self-collide, while the one including also the corrective torques will not. To reinforce this statement, some videos about the simulations with and without the learning correction are available at the GitHub repository linked in the references. However, this is not to say that the robot will self-collide in any case under the nominal controller or that the proposed learning algorithm can also avoid self-collisions, but it is, instead, that, under the assumption of a self-collision-free planning, there are higher probabilities for the robot to self-collide because of the dynamic uncertainties if it is controlled with just the nominal controller without the learning correction.

7.1 Future works

Further improvements on this topic, which can be subject of future works, can be achieved in different ways.

First, one may try the DLS solution for handling redundancy only when strictly needed. Indeed, if on one hand this solution provides robustness, on the other hand it introduces an error. Hence, by monitoring the smallest singular value of the robot Jacobian, it is possible to introduce the damping term only if it drops under a suitable threshold, switching it off otherwise, allowing to set this performance-robustness trade-off only when stricly needed.

Following, the control law may be extended by adding a robust term aimed to reduce the tracking error during the learning process. Indeed, it may happen to have some relatively large errors with respect to the desired task, and a robust component can be introduced in order to lower down the tracking error, benefiting the learning algorithm.

Moreover, to improve the computational time required by the algorithm, the fitting and the predictions of the n Gaussian processes may be processed in parallel instead of sequentially, resulting into a significant speed-up of the overall approach.

Finally, for real-time applications, e.g. an experiment on the real robot, the proposed software, which has been written in MATLAB, can be translated into C++ code.

Appendix A

Key Software

In this appendix the main MATLAB code used for implementing the proposed algorithm and performing the simulations is reported.
Moreover, for readability purposes, some lines have been cut off, simplified or split on more lines.
The structure of this appendix is organized as follows:

- in the first section, the main module is reported, representing the code used for the simulation of a KUKA LWR 4+ robot with 10% uncertainty plus unmodeled frictions as uncertain dynamics;
- in the second section, the implementation of a sparse linearized Gaussian process is proposed;
- in the third section, the Linear Quadratic optimization problem aimed at minimizing the variance in a computational efficient way is defined.

A.1 Main Module - Friction Case

```
%% KUKA LWR 4+ Robot - Simulation with both uncertainty on the masses
        and unmodeled frictions!

% choose trajectory
traj = 'sinusoidal';  % circular, sinusoidal, heart

% learning utilities
gp_flag = 1;                % {0,1} - set to 1 to use gp learning
opt_flag = 1;               % {0,1} - set to 1 to include
                                    null-space optimization
use_quadprog = 1;           % {0,1} - set to 1 to use quadprog
                                    function to optimize
                                    the variance, 0 to use fmincon
mod_value = 30;
lin_value = 2;              % set when upgrading the
                                    linearization point (iterative)
basis = 'constant';         % set the basis function the GP have to use
mean_flag = 1;              % (0 or 1) - set if you want to compute mu_hat
```

```
                                          as "...*Y" or "...*(Y-media)" ;
max_dim = 30;                % set the maximum dataset dimension
enable_check_collision = 0; % set to 1 to check for self-collisions
                             during the motion

%% Import robot by using URDF

% real robot
kuka = importrobot('./models/kuka_lwr.urdf');

% choose the uncertainty level on the  nominal robot

% perfect model
% kuka_nominal = kuka;

% uncertainties: +10%, +10%, -10%, -10%, +10%, -10%, +10%
% kuka_nominal = importrobot('./models/kuka_lwr_controller.urdf');

% uncertainties: -30%, +20%, -20%, -20%, +20%, -20%, +10%
% kuka_nominal = importrobot('./models/kuka_lwr_controller2.urdf');

% uncertainties: -60%, +40%, -50%, -30%, +20%, -20%, +20%
% kuka_nominal = importrobot('./models/kuka_lwr_controller3.urdf');

% uncertainties: -30%, +30%, -30%, -30%, +30%, -30%, +30%
kuka_nominal = importrobot('./models/kuka_lwr_controller4.urdf');

% setup kuka
kuka.Gravity = [0,0,-9.81];
kuka_nominal.Gravity = [0,0,-9.81];

kuka.DataFormat = 'row';
kuka_nominal.DataFormat = 'row';

%% Task definition

% time & timing law (row vector)
t0 = 0;         % initial time
Ts = 1e-2;      % simulation & control step
T = 5;          % total simulation time
t = t0:Ts:T;    % time
tau_t = t/T;    % normalized time

% cubic timing law, rest-to-rest motion
s = (-2*tau_t.^3 + 3*tau_t.^2);
ds = ((6*t)/T^2 - (6*t.^2)/T^3);
dds = (6/T^2 - (12*t)/T^3);

% desired Cartesian path hardly readable -> not reported

% inverse kinematics for setting up the first configuration
ik = inverseKinematics('RigidBodyTree', kuka);
weights = [0, 0, 0, 1, 1, 1]; % only position matters
```

```
endEffector = 'kuka_lwr_7_link';
qInitial = zeros(1,7); % initial guess
point =  p_ref(:,1);
[qSol, solInfo] = ik(endEffector,trvec2tform(point'),
            weights, qInitial);

%% initializations

% initial configuration - row vector
q0 = qSol;
q = zeros(length(t), 7);
q(1,:) = q0;

% initial velocity (at rest) - row vector
dq0 = zeros(1,7);
dq = zeros(length(t), 7);
dq(1,:) = dq0;

% initial acceleration - row vector
ddq0 = zeros(1,7);
ddq = zeros(length(t)-1, 7);
ddq(1,:) = ddq0;

% Jacobian
J = J_LWR(q(1),q(2),q(3),q(4),q(5),q(6));

% least square damping
damping = 0; % corresponds to the pseudo-inverse solution
% damping = 0.01;

% initial FL torque
TauFL = gravityTorque(kuka_nominal,q0);

% gains (cartesian PD)
Kp = diag([100, 100, 100]);
Kd = 2*sqrt(Kp);
Kv = 10;

% collisions
self_collision = [];

% remove collision cylinder on the EE (permanent self-collision)
clearCollision(kuka.Bodies{7});

% null-space optimization settings (nonlinear)
options_qp = optimoptions('quadprog', 'Display', 'off');
options_opt = optimoptions('fmincon','Algorithm','sqp',
        'Display', 'off');
A = [1;-1];
Aproj = blkdiag(A,A,A,A,A,A,A);
bproj = ones(14,1) * 10;
u0proj = zeros(7,1);
DeltaT = 1e-2;
```

```
% friction modeling, gains
alpha_f = diag([0.2 0.2 0.08 0.08 0.001 0.001 0.0001]);
beta_f = 0.2 * alpha_f;

% standard array structures
joints = [q0 dq0]; % configurations and velocities
singular_values = [];
task_vec = [];
accs_ref = [];
accs_redundant = [];
accs = [];
torques = [];
uopt = zeros(7,1);
friction = zeros(1,7);

% learning array structures
predictions = [];
variances = [];
Xdata = [];
Ydata = [];
Xdata_full = [];
Ydata_full = [];
corrective_torques = [];
needed_torques = [];
needed_torques_masses = [];
delta_eta = [];

%% Simulation
tic
for i = 1:length(t)-1
    disp(i*Ts)

    % Build up dataset
    if mod(i, lin_value) ==  0 || i-lin_value == 1
        if i > lin_value
            % sparse gp, keep only the last 'max_dim' points
%             if size(Xdata,1) > max_dim
%                 Xdata(1:end-max_dim+lin_value,:) = [];
%                 Ydata(1:end-max_dim+lin_value,:) = [];
%             end

            % compute dataset entries
            aux_X = [joints(end-lin_value:end-1,:),
                    accs_ref(end-lin_value+1:end,1:7)];
            aux_Y = -accs_ref(end-lin_value+1:end,1:7)
                    + accs(end-lin_value+1:end,1:7);

            % store entries in the dataset
            Xdata = vertcat(Xdata, aux_X);
            Ydata = vertcat(Ydata, aux_Y);
            Xdata_full = vertcat(Xdata_full, aux_X);
            Ydata_full = vertcat(Ydata_full, aux_Y);
        end

        % Tune GP's hyperparams
        if not(isempty(Xdata)) && (gp_flag == 1)
            % tune each mod_value step
            if mod(i,mod_value) == 0 || i-lin_value == 1

               if size(Xdata, 1) >= max_dim
                   gp_1 = fitrgp(Xdata,Ydata(:,1),
                            'FitMethod', 'sd',
                            'PredictMethod', 'sd',
                            'ActivesetSize', max_dim,
                            'ActiveSetMethod', 'sgma',
                            'KernelFunction','ardsquaredexponential');

                   % same up to gp_7

               else
                   gp_1 = fitrgp(Xdata,Ydata(:,1),
                            'KernelFunction','ardsquaredexponential');

                   % same up to gp_7

               end
               % select tuned hyperparameters
               if length(gp_1.KernelInformation.KernelParameters) == 22
                   sigma1 = gp_1.KernelInformation.KernelParameters;
                   sigma2 = gp_2.KernelInformation.KernelParameters;
                   sigma3 = gp_3.KernelInformation.KernelParameters;
                   sigma4 = gp_4.KernelInformation.KernelParameters;
                   sigma5 = gp_5.KernelInformation.KernelParameters;
                   sigma6 = gp_6.KernelInformation.KernelParameters;
                   sigma7 = gp_7.KernelInformation.KernelParameters;

                   sigmaL1 = sigma1(1:21)';
                   sigmaL2 = sigma2(1:21)';
                   sigmaL3 = sigma3(1:21)';
                   sigmaL4 = sigma4(1:21)';
                   sigmaL5 = sigma5(1:21)';
                   sigmaL6 = sigma6(1:21)';
                   sigmaL7 = sigma7(1:21)';

                   sigmaF1 = sigma1(22);
                   sigmaF2 = sigma2(22);
                   sigmaF3 = sigma3(22);
                   sigmaF4 = sigma4(22);
                   sigmaF5 = sigma5(22);
                   sigmaF6 = sigma6(22);
                   sigmaF7 = sigma7(22);

                   beta1 = gp_1.Beta;
                   beta2 = gp_2.Beta;
                   beta3 = gp_3.Beta;
```

```
                beta4 = gp_4.Beta;
                beta5 = gp_5.Beta;
                beta6 = gp_6.Beta;
                beta7 = gp_7.Beta;

                sigmaN1 = gp_1.Sigma;
                sigmaN2 = gp_2.Sigma;
                sigmaN3 = gp_3.Sigma;
                sigmaN4 = gp_4.Sigma;
                sigmaN5 = gp_5.Sigma;
                sigmaN6 = gp_6.Sigma;
                sigmaN7 = gp_7.Sigma;
             end
        else
           if i > max_dim
               gp_1 = fitrgp(Xdata,
                             Ydata(:,1),
                             'ActiveSetSize', max_dim,
                             'ActiveSetMethod', 'sgma',
                             'FitMethod', 'None',
                             'PredictMethod', 'sd',
                             'KernelFunction','ardsquaredexponential',
                             'KernelParameters',[sigmaL1';sigmaF1],
                             'BasisFunction', basis,
                             'Beta', beta1,
                             'Sigma', sigmaN1);

               % same up to gp_7

           else
               gp_1 = fitrgp(Xdata,
                         Ydata(:,1),
                         'ActiveSetSize', i,
                         'ActiveSetMethod', 'sgma',
                         'FitMethod', 'None',
                         'PredictMethod','sd',
                         'KernelFunction','ardsquaredexponential',
                         'KernelParameters',[sigmaL1';sigmaF1],
                         'BasisFunction', basis,
                         'Beta', beta1,
                         'Sigma', sigmaN1);

               % same up to gp_7

           end
        end
        Gps = {gp_1,gp_2,gp_3,gp_4,gp_5,gp_6,gp_7};
    end
end

% compute actual ee position
p = f(q(i,1), q(i,2), q(i,3), q(i,4), q(i,5), q(i,6));
```

```
% compute jacobian and its derivative
Jold = J;
J = J_LWR(q(i,1), q(i,2), q(i,3), q(i,4), q(i,5), q(i,6));
dJ = dJdt_LWR(dq(i,1), dq(i,2), dq(i,3), dq(i,4), dq(i,5), dq(i,6),
        q(i,1), q(i,2), q(i,3), q(i,4), q(i,5), q(i,6));

% check for singular values
[~,S,~] = svd(J);
singular_value = min(diag(S));

% actual ee velocity and acceleration
dp = J * dq(i,:)';
ddp = dJ * dq(i,:)' + J * ddq(i,:)';

% cartesian errors
err_p = p_ref(:,i) - p;
err_dp = dp_ref(:,i) - dp;

% acceleration reference
ddp_ref = Kp * err_p + Kd * err_dp + ddp_ff(:,i);

% projector
Pj = eye(7) - pinv(J) * J;

% reference task accelerations
A = J' * J + damping * eye(length(q(i,:)));
B = J' * (ddp_ref - dJ * dq(i,:)');
X1 = lsqminnorm(A,B);
Uref_task = X1';

% null-space optimization
if((gp_flag == 1) && (opt_flag == 1) && not(isempty(Xdata))
    && (length(gp_1.KernelInformation.KernelParameters) == 22))

    % set inputs to give to fmincon/quadprog
    beta = [beta1, beta2, beta3, beta4, beta5, beta6, beta7];
    sigmaN = [sigmaN1, sigmaN2, sigmaN3, sigmaN4,
            sigmaN5, sigmaN6, sigmaN7];

    % set linearization point
    if (mod(i,lin_value) == 0 || i-lin_value == 1)
        x_lin = Xdata(end,:);
    end

    if (use_quadprog == 0)  % use fmincon
        u_prev = uopt;
        [uopt,fval,exitflag,output] =
                fmincon(@(uproj)obj_proj_gp_lin(q(i,:), dq(i,:),
                uproj, Uref_task, DeltaT, Pj, u_prev, Kv, x_lin,
                Xdata, Ydata, sigma1, sigma2, sigma3, sigma4,
                sigma5, sigma6, sigma7, beta, sigmaN),
                u0proj, Aproj, bproj, [], [], [], [],
                [], options_opt);
        u0proj = uopt;
    else                    % use quadprog
        [H, f_] = quadratic_problem_2(q(i,:), dq(i,:),
                DeltaT, x_lin, Xdata, Ydata,
                sigma1, sigma2, sigma3, sigma4,
                sigma5, sigma6, sigma7, beta, sigmaN);
        delta_ddq = quadprog((H+H')/2, f_, Aproj, bproj,
                [], [], [], [], [], options_qp);
        uopt = delta_ddq;
    end

else
    uopt = zeros(7,1);
end

% redundancy resolution: null-space optimization
Uref_redundant = (Pj * (uopt - Kv*dq(i,:)') )';

% total reference acceleration
Uref = Uref_task + Uref_redundant;

% GP predictions
if(not(isempty(Xdata)) && (gp_flag == 1)
    && length(gp_1.KernelInformation.KernelParameters) == 22 )

    % upgrade and set linearization point
    if (mod(i,lin_value) == 0 || i-lin_value == 1)
        x_lin = Xdata(end,:);
    end

    % call the linearized RGP
    [mu_lin1, var_lin1] =
            linearizedRGP_sparse(gp_1.ActiveSetVectors,
                x_lin, gp_1.Alpha, sigmaL1,
                sigmaF1, beta1, sigmaN1);

    % same up to [mu_lin7, var_lin7]

    % compute dx and x_hat
    dx = [q(i,:), dq(i,:), Uref] - x_lin; % delta x
    x_hat = [1; dx']; % x_hat

    % compute predictions mean
    new_mu1 = mu_lin1'*x_hat + beta1;
    new_mu2 = mu_lin2'*x_hat + beta2;
    new_mu3 = mu_lin3'*x_hat + beta3;
    new_mu4 = mu_lin4'*x_hat + beta4;
    new_mu5 = mu_lin5'*x_hat + beta5;
    new_mu6 = mu_lin6'*x_hat + beta6;
    new_mu7 = mu_lin7'*x_hat + beta7;

    % compute predictions variance
    new_var1 = x_hat' * var_lin1 * x_hat;
    new_var2 = x_hat' * var_lin2 * x_hat;
    new_var3 = x_hat' * var_lin3 * x_hat;
    new_var4 = x_hat' * var_lin4 * x_hat;
    new_var5 = x_hat' * var_lin5 * x_hat;
    new_var6 = x_hat' * var_lin6 * x_hat;
    new_var7 = x_hat' * var_lin7 * x_hat;

    % regroup predictions and variances
    prediction_vector = [new_mu1, new_mu2, new_mu3, new_mu4,
            new_mu5, new_mu6, new_mu7];
    variance_vector = [new_var1, new_var2, new_var3, new_var4,
            new_var5, new_var6, new_var7];

    % subtract the GP prediction to the reference acceleration
    % the - sign is according to how the dataset has been built
    Uref = Uref - prediction_vector;
else
    prediction_vector = zeros(1,7);
    variance_vector  = zeros(1,7);
end

% check for self collisions
if enable_check_collision == 1 && i > 1
    self_collision(i) = checkCollision(kuka, q(i,:));
end
check = any(self_collision == 1);

if check == 0
    % Nominal control law: Computed Torque (FL + PD + FFW)
    TauFL = (massMatrix(kuka_nominal, q(i,:)) * Uref')'
            + velocityProduct(kuka_nominal,q(i,:),dq(i,:))
            + gravityTorque(kuka_nominal,q(i,:));
else
    % Collision detected!
    disp('Self␣collision␣detected!')
    break;
end

% compute friction
friction(i,:) = 0.5*(alpha_f*(1./(1+exp(-30*dq(i,:))))'
        - 1/2*ones(7,1) + beta_f*dq(i,:)')';

% dynamic model inversion - row vector
ddq(i,:) = inv(massMatrix(kuka,q(i,:))) * (TauFL
        - velocityProduct(kuka,q(i,:),dq(i,:))
        - gravityTorque(kuka,q(i,:))
        - friction(i,:) )';

% numeric integration (Euler method)
dq(i+1,:) = dq(i,:) + ddq(i,:) * Ts;
q(i+1,:) = q(i,:) + dq(i,:) * Ts;

% Compute DeltaM and DeltaNi
DeltaM = massMatrix(kuka,q(i,:))
        - massMatrix(kuka_nominal,q(i,:));
DeltaNi =  velocityProduct(kuka,q(i,:),dq(i,:))
        + gravityTorque(kuka,q(i,:))
        + friction(i,:)
        - velocityProduct(kuka_nominal,q(i,:),dq(i,:))
        - gravityTorque(kuka_nominal,q(i,:));

% Corrective Torque
Torque_corrective = (-massMatrix(kuka_nominal, q(i,:))
                * prediction_vector')';

% Real torque disturbance acting on the system
Torque_needed = (DeltaM * ddq(i,:)' + DeltaNi')';
Torque_needed_masses = (DeltaM * ddq(i,:)')';

% update standard array structures
task_vec = vertcat(task_vec, [p',dp',ddp',p_ref(:,i)',
        dp_ref(:,i)',ddp_ref']);
accs_ref = vertcat(accs_ref, Uref);
accs_redundant = vertcat(accs_redundant, uopt');
accs = vertcat(accs, ddq(i,:));
singular_values = vertcat(singular_values, singular_value);
joints = vertcat(joints, [q(i+1,:), dq(i+1,:)]);

% update learning array structures
predictions = vertcat(predictions, prediction_vector);
variances = vertcat(variances, variance_vector);
corrective_torques = vertcat(corrective_torques,
        Torque_corrective);
needed_torques = vertcat(needed_torques, Torque_needed);
needed_torques_masses = vertcat(needed_torques_masses,
        Torque_needed_masses);
delta_eta = vertcat(delta_eta, DeltaNi);
torques = vertcat(torques,TauFL);

end
toc
```

A.2 Sparse Linearized Gaussian Process

```
function [mu_lin, var_lin] = linearizedRGP_sparse(Xdata, x_lin,
        Ydata, sigmaL, sigmaF, beta, sigma_N)

    % function handles
    my_covariances = @my_covariances_extended;

    % compute numeric covariances
    [K, K_, K__] = my_covariances(Xdata, x_lin, sigmaL, sigmaF);

    % adding noise
    K = K + sigma_N^2 * eye(size(K,1));
    K__ = K__ + sigma_N^2;

    % inverse of K (efficient computation)
    Kinv = K \ eye(size(K,1));

    % compute all the derivatives
    K_01_Xx = K_.*(Xdata-x_lin)./(sigmaL.^2);
    K_10_xX = - K_' .* (Xdata'-x_lin')./(sigmaL'.^2);
    K_01_xx = zeros(1,21);
    K_10_xx = zeros(21,1);
    K_11_xx = diag(sigmaF^2./sigmaL.^2);

    % basis function
    H = ones(size(Xdata,1),1); % with constant basis function
    media = H*beta;

    % compute linearized mean and variance
    mu_lin = [K_';  K_10_xX] * Ydata;
    var_lin = [K__, K_01_xx; K_10_xx, K_11_xx]
            - [K_'; K_10_xX] * Kinv * [K_, K_01_Xx];

end
```

A.3 LQ Variance Optimization Problem

```
%% LQ variance optimization by splitting time t from time t+1

function [H, f] = quadratic_problem_2(q_input, dq_input, DeltaT,
            x_lin, Xdata, Ydata, sigma1, sigma2, sigma3, sigma4,
            sigma5, sigma6, sigma7, beta, sigmaN)

    % define variables
    sigmaL1 = sigma1(1:21)';
    sigmaL2 = sigma2(1:21)';
    sigmaL3 = sigma3(1:21)';
    sigmaL4 = sigma4(1:21)';
    sigmaL5 = sigma5(1:21)';
    sigmaL6 = sigma6(1:21)';
    sigmaL7 = sigma7(1:21)';

    sigmaF1 = sigma1(22);
    sigmaF2 = sigma2(22);
    sigmaF3 = sigma3(22);
    sigmaF4 = sigma4(22);
    sigmaF5 = sigma5(22);
    sigmaF6 = sigma6(22);
    sigmaF7 = sigma7(22);

    beta1 = beta(1);
    beta2 = beta(2);
```

```
beta3 = beta(3);
beta4 = beta(4);
beta5 = beta(5);
beta6 = beta(6);
beta7 = beta(7);

sigmaN1 = sigmaN(1);
sigmaN2 = sigmaN(2);
sigmaN3 = sigmaN(3);
sigmaN4 = sigmaN(4);
sigmaN5 = sigmaN(5);
sigmaN6 = sigmaN(6);
sigmaN7 = sigmaN(7);

N = length(q_input);

% Compute linearized variances
[~, var_lin1] = linearizedRGP(Xdata, x_lin, Ydata(:,1),
        sigmaL1, sigmaF1, beta1, 1, sigmaN1);
[~, var_lin2] = linearizedRGP(Xdata, x_lin, Ydata(:,2),
        sigmaL2, sigmaF2, beta2, 1, sigmaN2);
[~, var_lin3] = linearizedRGP(Xdata, x_lin, Ydata(:,3),
        sigmaL3, sigmaF3, beta3, 1, sigmaN3);
[~, var_lin4] = linearizedRGP(Xdata, x_lin, Ydata(:,4),
        sigmaL4, sigmaF4, beta4, 1, sigmaN4);
[~, var_lin5] = linearizedRGP(Xdata, x_lin, Ydata(:,5),
        sigmaL5, sigmaF5, beta5, 1, sigmaN5);
[~, var_lin6] = linearizedRGP(Xdata, x_lin, Ydata(:,6),
        sigmaL6, sigmaF6, beta6, 1, sigmaN6);
[~, var_lin7] = linearizedRGP(Xdata, x_lin, Ydata(:,7),
        sigmaL7, sigmaF7, beta7, 1, sigmaN7);

% sum of the variances
V = var_lin1 + var_lin2 + var_lin3 + var_lin4 + var_lin5
    + var_lin6 + var_lin7;

% minimizing variance at time t
bt = [1;
    (q_input - x_lin(1:7))';
    (dq_input - x_lin(8:14))';
    (-x_lin(15:21))'];
At = [zeros(1,7);
    zeros(7,7);
    zeros(7,7);
    eye(7,7)];
Ht = 2*At'*V*At;
ft = (2*bt'*V*At)';

% minimizing variance at time t+1
b_next = [1;
        (q_input - x_lin(1:7) + dq_input*DeltaT)';
        (dq_input - x_lin(8:14))';
        (-x_lin(15:21))'];
    A_next = [zeros(1,N);
            eye(N)*DeltaT^2;
            eye(N)*DeltaT;
            zeros(N)];
    H_next = A_next' * (2*V) * A_next;
    f_next = (2*b_next'*V*A_next)';

    % recall:   - quadratic problem are w.r.t ddq_t,
    %               not x_hat_t and x_hat_t+1
    %           - the weights may change depending on the trajectory
    H = 1*Ht + 5*H_next;
    f = 1*ft + 5*f_next;
end
```

Bibliography

[1] M. Ficorilli, “On-line learning control of redundant robots with uncertain dynamics.” `https://github.com/marcoficorilli/master-thesis`, 2022.

[2] 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. 2046–2057, 2019.

[3] F. Caccavale, S. Chiaverini, and B. Siciliano, “Second-order kinematic control of robot manipulators with Jacobian damped least-squares inverse: Theory and experiments,” *IEEE/ASME Transactions on Mechatronics*, vol. 2, no. 3, pp. 188–194, 1997.

[4] 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 *3rd Conference on Robot Learning*, vol. 100, pp. 1359–1368, Proc. of Machine Learning Research, 2020.

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

[6] 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.

[7] T. X. Nghiem, T.-D. Nguyen, and V.-A. Le, “Fast Gaussian process based model predictive control with uncertainty propagation,” in *57th Annual Allerton Conference on Communication, Control, and Computing*, pp. 1052–1059, 2019.

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

[9] C. E. Rasmussen, “Gaussian processes in machine learning,” in *Advanced Lectures on Machine Learning* (O. Bousquet, U. von Luxburg, and G. Rätsch, eds.), vol. 317 of *Lecture Notes in Computer Science*, pp. 63–71, Springer, 2003.

[10] L. Sciavicco, B. Siciliano, L. Villani, and G. Oriolo, “Robotics: Modeling, Planning and Control.” Springer, 2011.

[11] G. Turrisi, M. Capotondi, C. Gaz, V. Modugno, G. Oriolo, and A. De Luca, "On-line learning for planning and control of underactuated robots with uncertain dynamics," *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 358–365, 2021.

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