Elective in Robotics – Underactuated Robots

FP7. Robot learning techniques for set point regulation of robots with nonlinear flexibility on the joints

A.Caciolai E.Nicotra M.Rabbiolo

Dipartimento di Ingegneria Informatica Automatica e Gestionale
Antonio Ruberti
Control Engineering – Artificial Intelligence and Robotics
Sapienza Università di Roma

Academic year 2020/2021

Professor L. Lanari & G. Oriolo
Supervisor M. Capotondi

Contents

1 Introduction

Set-point regulation for flexible joints robot is a classic control problem in underactuated robotics.

In fact, the noncollocation of the torque inputs and the elastic coupling between the motor and the link makes this kind of system very challenging to control, with respect to the rigid ones. In spite of this, many techniques were developed for realizing set-point regulation, in most cases model-based such as static/dynamic feedback linearization or backstepping, obtaining great performance at the expanse of high precision in the system modeling. The purpose of this project is to study how a data-driven control technique can be applied when the system model is not precisely known, exploiting a robot learning method developed at DIAG, analyzing its applicability and performance in this context. In particular, the latter is used to estimate the nonlinear elastic term of the robot model, an approach similar the one presented in [1][2].

2 Robot with flexible joints

In industrial robots the presence of transmission elements such as harmonic drives and long shafts introduces flexibility effects between the actuating inputs (motors) and driven outputs (links). Also, recently, flexible actuation/transmission elements have been deliberately selected in robots intended for physical human–robot interaction [3]. These phenomena are captured by modeling the *flexibility at the robot joints*.

In this project we have used the so-called *Spong model*, that has become a widely used standard model for robots with elastic joints. The model is derived with the familiar Lagrangian approach, under a certain set of assumptions.

2.1 Spong model

Consider an open-chain robot with N revolute *elastic* joints and N *rigid* links. Because of the additional degrees of freedom introduced by the elastic coupling of the motor shaft to the links, one models the motor of each actuator as a "fictitious link," that is, as an additional rigid body in the chain with its own inertia. Therefore, we will have $2N$ moving rigid bodies (links and motors) in the robot chain, and accordingly, $2N$ generalized coordinates will be needed. [4][3] These variables can be grouped into *motor variables* $\boldsymbol{\theta} \in \mathbb{R}^N$ and *link variables* $\mathbf{q} \in \mathbb{R}^N$.

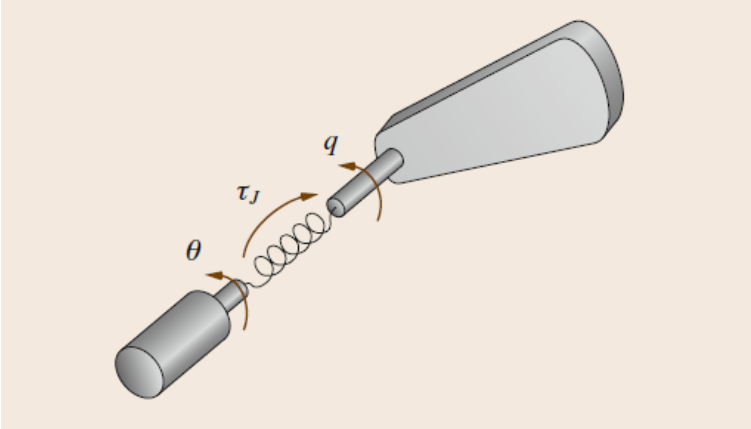


Figure 1: Schematic representation of an elastic joint

The assumptions that underlie the Spong model are the following [3]:

1. Joint deflections are small, so that flexibility effects are limited to the domain of linear elasticity.
2. The actuators' rotors are modeled as uniform bodies having their center of mass on the rotation axis.
3. Each motor is located on the robot arm in a position preceding the driven link, so that the inertia matrix attains a simplified structure.
4. The kinetic energy of the rotors is mainly due to their own rotation, an assumption that leads to the full inertial decoupling between rotor and link variables.

Actually, in our project we have assumed a general nonlinear elastic term, therefore dropping the first assumption.

Following a Lagrangian approach, the single energy contributions to the Lagrangian

$$\mathcal{L} = \mathcal{T}(\mathbf{q}, \dot{\mathbf{q}}) - \mathcal{U}(\mathbf{q}) \tag{1}$$

will be derived next.

The *potential energy* $\mathcal{U}(\mathbf{q})$ of the robot is due to gravity and joint elasticity. The gravity part U_g is related to the position of the barycenter of the links (each of mass m_i) and of the motors (of mass m_{r_i}). Because of assumption 2, the latter will be independent of $\boldsymbol{\theta}$. Thus

$$\mathcal{U}_g = \mathcal{U}_{g,l}(\mathbf{q}) + \mathcal{U}_{g,m}(\mathbf{q}). \tag{2}$$

The elastic part $\mathcal{U}_e$ is a function of the of the displacement $\mathbf{q} - \theta$, and depends on the chosen elastic term $\boldsymbol{\psi}(\mathbf{q} - \boldsymbol{\theta})$, since physically elastic force is a conservative force and therefore it is the negative gradient of a potential:

$$\boldsymbol{\psi}(\mathbf{q} - \boldsymbol{\theta}) = -\boldsymbol{\nabla} U_e. \tag{3}$$

A physical requirement for the elastic potential is the symmetry, i.e. $\mathcal{U}_e(\mathbf{q} - \boldsymbol{\theta}) = \mathcal{U}_e(\boldsymbol{\theta} - \mathbf{q})$, so that for instance if one chooses a polynomial to represent the elastic term, one must choose an odd polynomial so that the corresponding potential is an even polynomial, hence a symmetric function.

The *kinetic energy* of the robot is the sum of the link and motor contributions. For the links, there is no difference with respect to the standard rigid robot case and it will be sufficient to write in general

$$\mathcal{T}_l = \frac{1}{2} \dot{\mathbf{q}}^T \mathbf{M}_l(\mathbf{q}) \dot{\mathbf{q}} \tag{4}$$

where $\mathbf{M}_l(\mathbf{q})$ is the positive-definite symmetric link inertia matrix.

For the motors, we distinguish between linear and angular contributions. The motor linear kinetic energy $\mathcal{T}_{m,L}$ can be modeled as just an additional mass of the carrying link, and since the spinning of the motor does not affect the linear kinetic energy of the motor, only its rotational kinetic energy, we can group the two terms up as follows

$$\mathcal{T}_l + \mathcal{T}_{m,L} = \frac{1}{2} \dot{\mathbf{q}}^T (\mathbf{M}_l(\mathbf{q}) + \mathbf{M}_m(\mathbf{q})) \dot{\mathbf{q}} = \frac{1}{2} \dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \dot{\mathbf{q}}, \tag{5}$$

where $\mathbf{M_m}(\mathbf{q})$ contains the motor masses (and, possibly, the motor inertial components along the other principal axes).

The angular kinetic energy of each motor, expressed in a reference frame attached to said motor, attains a simple form thanks to assumption 4, as anticipated above. In fact, we have

$$\boldsymbol{\omega}_{r_i} = \begin{pmatrix} 0 & 0 & \dot{\theta}_{m,i} \end{pmatrix} \tag{6}$$

without any dependence on the link variables. Basically, this means neglecting the contribution of the previous links' angular velocities, that is

$$\sum_{j=1}^{i-1} \mathbf{J}_{r_i,j}(\mathbf{q}) \dot{q}_j \tag{7}$$

where $\mathbf{J}_{r_i,j}$ is the j-th column of the Jacobian relating the link velocities $\dot{q}$ to the angular velocity of the i-th motor in the robot chain. This is a reasonable assumption since it amounts to neglecting

terms of order at most $1/g_r$ [4], where g_r is the gear ratio, that can be as high as 200 in the case of harmonic drives. So, the motor rotational kinetic energy is

$$\mathcal{T}_{m,A} = \sum_{i=1}^{N} \frac{1}{2} \boldsymbol{\omega}_{r_i}^T \mathbf{I}_{r_i} \boldsymbol{\omega}_{r_i} = \frac{1}{2} \dot{\boldsymbol{\theta}}^T \mathbf{B} \dot{\boldsymbol{\theta}} \tag{8}$$

where $\mathbf{B}$ is the constant diagonal inertia matrix collecting the motors inertial components $I_{r_i,zz}$ around their spinning axes.

Now, we can compute the Lagrangian of the system:

$$\mathcal{L} = \mathcal{T} - \mathcal{U} = (\mathcal{T}_l + \mathcal{T}_{m,L} + \mathcal{T}_{m,A}) - (\mathcal{U}_g + \mathcal{U}_e) \tag{9}$$

$$= \frac{1}{2} \dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \dot{\mathbf{q}} + \frac{1}{2} \dot{\boldsymbol{\theta}}^T \mathbf{B} \dot{\boldsymbol{\theta}} - \mathcal{U}_g(\mathbf{q}) - \mathcal{U}_e(\mathbf{q} - \boldsymbol{\theta}), \tag{10}$$

and obtain the mathematical formulation of Spong model using the Euler-Lagrange equations. Let $\mathbf{p} = \begin{pmatrix} \mathbf{q}^T & \boldsymbol{\theta}^T \end{pmatrix}$, and $\mathbf{u} \in \mathbb{R}^{2N}$ the non-conservative generalized torques performing work on $\mathbf{p}$, then the equations

$$\frac{\mathrm{d}}{\mathrm{d}t} \left(\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{p}}} \right)^T - \left(\frac{\partial \mathcal{L}}{\partial \mathbf{p}} \right)^T = \mathbf{u} \tag{11}$$

lead to the dynamical model

$$\mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) + \boldsymbol{\psi}(\mathbf{q} - \boldsymbol{\theta}) + \mathbf{D} \dot{\mathbf{q}} = \mathbf{0} \tag{12}$$

$$\mathbf{B} \ddot{\boldsymbol{\theta}} - \boldsymbol{\psi}(\mathbf{q} - \boldsymbol{\theta}) + \mathbf{D} \dot{\boldsymbol{\theta}} = \boldsymbol{\tau} \tag{13}$$

in which

- $\mathbf{c}(\mathbf{q}, \dot{\mathbf{q}})$ collects the Coriolis and centrifugal terms.
- $\mathbf{g}(\mathbf{q})$ is the gravity term ($= \partial \mathcal{U}_g / \partial q$).
- $\boldsymbol{\tau} \in \mathbb{R}^N$ is the vector of generalized torques performing work on $\boldsymbol{\theta}$.
- $\mathbf{D}$ is a damping coefficient.

2.2 Underactuated robot

The model described in the previous section is one of an *underactuated robot*. Before explaining why that is the case, let us briefly summarize what an underactuated robot is.

For a generic robot, we define its *configuration* as a minimal set of *generalized coordinates* that can uniquely describe positions of all points of the robot, in an appropriate frame. A configuration associates to each posture of the robot a point in a properly defined *configuration space* $\mathcal{C}$, i.e. the set of all configurations that the robot can assume. Since generalized coordinates can be Cartesian or angular, in general this is not an Euclidean space, but a *manifold*. Nevertheless, one usually describes a configuration as a vector $\mathbf{q} \in \mathbb{R}^n$, but this description is valid only locally. Another useful definition is that of *degree of freedom* (*dof* in short), defined as the number of *independent displacements* (velocities) possible in the configuration space $\mathcal{C}$. For an *unconstrained* (or *free-flying*) robot, the number m of degrees of freedom and the number n of generalized coordinates coincides;

on the other hand, if the robot is subject to k constraints, then the number of dof's decreases accordingly, i.e. $m = n - k$. [5]

With these definitions, we can define an underactuated robot as a robot with fewer control inputs than degrees of freedom [6] [7]. To formalize this definition, consider a general unconstrained mechanical system, described by a nonlinear state-space model in the general form:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \sum_{i=1}^{p} \mathbf{g}_i(\mathbf{x}) u_i = \mathbf{f}(\mathbf{x}) + \mathbf{G}(\mathbf{x})\mathbf{u} \tag{14}$$

where the state $\mathbf{x}$ usually comprises "position" and "velocity", i.e. $\mathbf{x} = \begin{pmatrix} \mathbf{q}^T & \dot{\mathbf{q}}^T \end{pmatrix}^T \in \mathbb{R}^{2n}$, the term $\mathbf{f}(\mathbf{x})$ is the *drift* vector field, the terms $\mathbf{g}_i(\mathbf{x})$ are the *input* vector fields, and u_i are the control inputs. The system described by eq. (14) is said to be underactuated if $p < m = n$, i.e. if $rank(\mathbf{G}) < dim(\mathbf{x})$ [8].

After the digression above, let us verify that a robot with elastic joints is indeed an underactuated robot. If we look at eqs. (12) and (13) the control input can only affect directly the motor variables, which in turn are dynamically coupled to the link variables through the elastic term $\boldsymbol{\psi}$. This observation is supported by the theory: the configuration space $\mathcal{C} \ni \begin{pmatrix} \mathbf{q}^T & \boldsymbol{\theta}^T \end{pmatrix}^T$ is $2n$-dimensional, and the system is unconstrained so we have $2n$ degrees of freedom, while the input space is only n-dimensional, therefore we have indeed fewer control inputs than degrees of freedoms and we can conclude that the robot is underactuated, according to the definition above.

However, the system under consideration is not a generic underactuated system, but a rather simple one, for which the problems of trajectory planning, trajectory tracking and set-point regulation are solvable [7]. In fact, Spong himself addresses the simplicity of his model in his paper [4], where he shows that the system is *feedback linearizable* and thus linearly controllable, using $\mathbf{y} = \mathbf{q}$ as linearizing coordinates. Indeed, the system can be described in state-space with a $4n$-dimensional state $\mathbf{x}^T = \begin{pmatrix} \mathbf{q}^T & \boldsymbol{\theta}^T & \dot{\mathbf{q}}^T & \dot{\boldsymbol{\theta}}^T \end{pmatrix}^T$ and it can be shown that it has relative degree exactly equal to $4n$ with respect to $\mathbf{y}$, therefore it is feedback linearizable [9]. Therefore, we have theoretical guarantees that the task of set-point regulation that we set out to accomplish in this project is indeed realizable.

3 MPC

In our project we relied on *Model Predictive Control* (MPC) to compute the input torques to eqs. (12) and (13), since this approach can provide smooth inputs while satisfying a desired set of *constraints*. The MPC framework is an open loop control scheme in which in the current control action is obtained by solving at each generic instant of time k a *finite horizon optimal control problem* using the current state of the plant as initial state. The optimal solution consists in a sequence of control inputs that satisfy the constraints while minimizing a user-specified cost function over a finite time window of N steps in the future. Only the first control of the sequence is applied to the plant. The remaining sequence is discarded and the process is iterated at time $k+1$ (see fig. 2 for a graphical representation of this scheme).

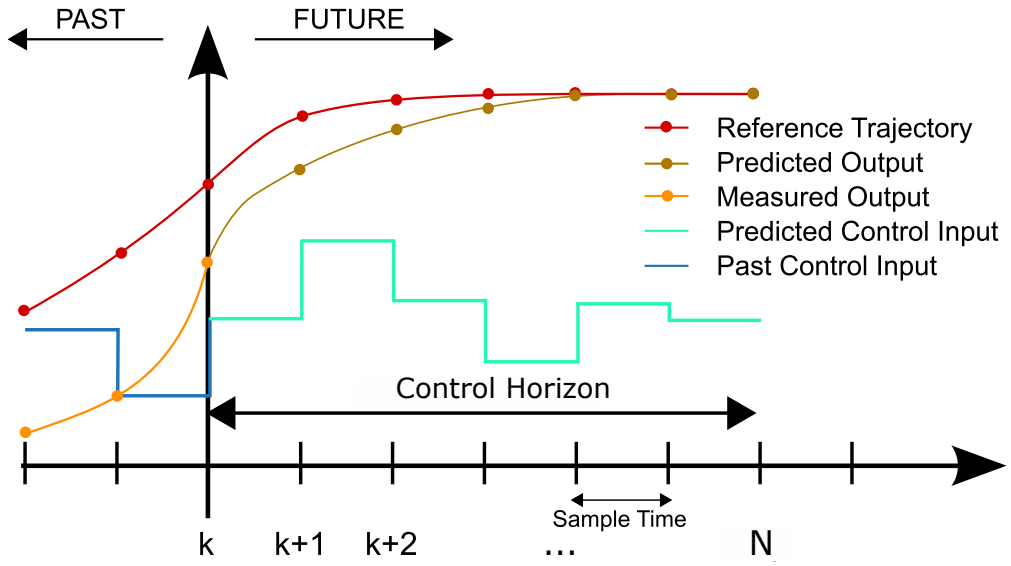


Figure 2: MPC scheme

As shown in section 2, the robot model consists of two (coupled) sets of $n = 2$ differential equations, where n corresponds to the number of joints, with each joint receiving an input torque $u^j = \tau^j$ with $j = 1, 2$. Notice that these equations are nonlinear, therefore we must resort to *nonlinear* MPC. In this formulation, MPC attains the following form [10]:

$$\min_{\mathbf{u}} V(\mathbf{x}, \mathbf{u}) \tag{15}$$

$$\text{s.t. } \mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k)) \tag{16}$$

$$\mathbf{h}(\mathbf{x}, \mathbf{u}) \leq 0 \tag{17}$$

in which $V : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is the (possibly nonlinear) *cost function*, $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is the *prediction model* that allows MPC to internally simulate the dynamics of the controlled system in order to check the optimality of the issued commands according to the simulated system response, and $\mathbf{h} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^q$ are the q (possibly nonlinear) constraints.

3.1 MPC Constraints and Cost Function

The MPC optimization problem is solved by minimizing the cost function defined as

$$V(\mathbf{x}, \mathbf{u}) = \sum_{i=0}^{N-1} l(\mathbf{x}(i), \mathbf{u}(i)) \tag{18}$$

where $\mathbf{u} = \{\mathbf{u}(0), \mathbf{u}(1), ..., \mathbf{u}(N-1)\}$ is the control sequence over the control horizon N and $\mathbf{x}(i)$ is the state at time $i = 1, ..., N-1$ [10]. In particular, the stage cost $\ell(\cdot)$ is defined as the sum of a regulation cost J_t, a velocity damping cost $J_{\dot{\boldsymbol{\theta}}}$ penalizing large values of $\dot{\boldsymbol{\theta}}$ and finally, a smoothness cost J_s penalizing large variations in u^j:

$$V(\mathbf{x}, \mathbf{u}) = \sum_{i=0}^{N-1} \ell(\mathbf{x}(i), \mathbf{u}(i)) = \alpha_1 J_t + \alpha_2 J_{\dot{\theta}} + \alpha_3 J_s \tag{19}$$

where

$$\begin{aligned} J_t &= \sum_{i=0}^{N-1} (\theta_{d,i} - \theta_i)^T (\theta_{d,i} - \theta_i) \\ J_{\dot{\theta}} &= \sum_{i=0}^{N-1} (\dot{\theta}_i)^2 \\ J_s &= \sum_{i=0}^{N-1} (\mathbf{u}_{i+1} - \mathbf{u}_i)^T (\mathbf{u}_{i+1} - \mathbf{u}_i) \end{aligned}$$

The feasibility of the commanded torques is obtained by imposing on the optimization problem the constraint

$$\boldsymbol{\tau}_m \leq \mathbf{u} \leq \boldsymbol{\tau}_M \tag{20}$$

where $\boldsymbol{\tau}_m$ and $\boldsymbol{\tau}_M$ are respectively a lower and upper bound on the torques. Minimization of (18) yields the desired control sequence

$$\mathbf{u}^* = \{\mathbf{u}^*(0), \mathbf{u}^*(1), ..., \mathbf{u}^*(N-1)\} \tag{21}$$

of which, as anticipated above, only the first one is actually fed to the system.

3.2 MPC Stability Issues

The control law $\mathbf{u}^*$ obtained in section section 3.1 is not necessarily optimal nor stabilizing since the optimization problem solved by the MPC is finite horizon. However, it has been proved [11][10] that closed loop stability can be solved by adding to eq. (18) a terminal cost $V_f(\cdot)$ which is a local control Lyapunov function to the objective function eq. (18) in a neighbourhood of the desired state:

$$V(\mathbf{x}, \mathbf{u}) = \sum_{i=0}^{N-1} \ell(\mathbf{x}(i), \mathbf{u}(i)) + \sum_{k=N-1-K}^{N-1} V_f(\mathbf{x}(k), \mathbf{u}(k)) \tag{22}$$

with $i = 1, .., N-1$ and $k = N-1-K, ..., N-1$.

Thus, the optimization problem is modified by the addition of a terminal cost $V_f(\cdot)$ and a terminal constraint applied on the last K steps of the control horizon N which is defined as

$$|\boldsymbol{\theta}(k) - \boldsymbol{\theta}_d(k)| \leq \epsilon \quad \text{with} \quad k = N-1-K, ..., N-1 \tag{23}$$

with ϵ defining the side of an arbitrarily small "square" in the subspace of configuration space pertaining to motor variables centered around the desired value for the motor variables.

In particular, $V_f(\cdot)$ is chosen as the terminal cost of the constrained linear-quadratic regulator (LQR) computed on the linearized system. Indeed, the LQR is converted to a finite horizon form as

$$J(\mathbf{u}, t) = \sum_{i=1}^{N-1} [\mathbf{x}_{t+i}^T \mathbf{Q} \mathbf{x}_{t+i} + \mathbf{x}_{t+i-1}^T \mathbf{R} \mathbf{u}_{k+i-1}] + \mathbf{x}_{t+N}^T \mathbf{S} \mathbf{x}_{t+N} \tag{24}$$

where $\mathbf{Q}, \mathbf{R}$ are symmetric positive semi-definite weights. Matrix $\mathbf{S}$ is a symmetric positive semi-definite terminal penalty weight applied at the final prediction horizon step and solution to the Riccati equation

$$\mathbf{S} = \mathbf{A}^T \mathbf{S} \mathbf{A} - \mathbf{A}^T \mathbf{S} \mathbf{B} (\mathbf{B}^T \mathbf{S} \mathbf{B} + \mathbf{R})^{-1} \mathbf{B}^T \mathbf{S} \mathbf{A} + \mathbf{Q} \tag{25}$$

where $\mathbf{A}$ and $\mathbf{B}$ are the matrices characterizing the model in state-space form.
So, in our framework assuming that $\mathbf{x}_N \sim \mathbf{x}_d$ we defined J_{LQR} as

$$J_{LQR} = \mathbf{x}_d^T \mathbf{S} \mathbf{x}_d \tag{26}$$

where matrix $\mathbf{S}$ is computed from eq. (25) and matrices $\mathbf{A}$ and $\mathbf{B}$ are obtained by linearizing the model eqs. (12) and (13) in the desired state. The addition of such terminal cost J_{LQR} ensures local convergence to the desired equilibrium state $\mathbf{x}_d$

Furthermore, to increase stability at each iteration a gravity compensation term is added to the optimal input torque $\boldsymbol{\tau}^*$ computed by the MPC, as suggested in [12]. As stated in [13], the input torque should satisfy

$$\boldsymbol{\tau}_g = \mathbf{g}(\mathbf{q_d}) \tag{27}$$

where τ_g denotes the gravity compensation term. Thus, at each iteration, the control input τ^* is updated as

$$\boldsymbol{\tau}^* = \boldsymbol{\tau}_{MPC}^* + \boldsymbol{\tau}_g \tag{28}$$

The predictive model of MPC explicity accounts for this compensation, indeed

$$\hat{\mathbf{x}}(i+1) = \mathbf{f}(\hat{\mathbf{x}}(i), \boldsymbol{\tau}_{MPC}^*(i) + \boldsymbol{\tau}_g) \tag{29}$$

As the system reaches the equilibrium, the optimal control input $\boldsymbol{\tau}_{MPC}^*$ computed by MPC goes to zero.

4 Learning techniques

In real case scenario the elastic term introduced in ref(???) is almost never known exactly. In particular, its nonlinearity feature makes the adoption of a learning technique necessary, in order to build a learning model which is able to predict its value at each iteration.
In this project two different learning methods are presented and compared.
Before going into the details of the two models, it is important to define the dataset upon which the training step is performed and its collection procedure.
As it has been said in the previous sections the elastic term depends on the difference between the motor variables *motor variables* $\boldsymbol{\theta} \in \mathbb{R}^N$ and the *link variables* $\mathbf{q} \in \mathbb{R}^N$. Then it is reasonable to take $\mathbf{X} = [\mathbf{q} - \boldsymbol{\theta}] \in \mathbb{R}^N$ as input and the elastic term $\mathbf{Y} = \boldsymbol{\psi}(\mathbf{q} - \boldsymbol{\theta}) \in \mathbb{R}^N$ as output.
Assuming that $\mathbf{q}, \boldsymbol{\theta}, \dot{\mathbf{q}}$ and $\dot{\boldsymbol{\theta}}$ are measurable, adding a point to the input set is straightforward.
For what concerns the output, from eq. (13) it is easy to obtain

$$\boldsymbol{\psi}(\mathbf{q} - \boldsymbol{\theta}) = \boldsymbol{\tau} - \mathbf{B}\ddot{\boldsymbol{\theta}} \tag{30}$$

So in order to obtain the elastic term, the motors' acceleration is needed. The latter is achieved through a numerical derivation, namely the Euler's rule.

$$\ddot{\boldsymbol{\theta}}_k = \frac{\dot{\boldsymbol{\theta}}_k - \dot{\boldsymbol{\theta}}_{k-1}}{T} \tag{31}$$

In conclusion by driving the robot on "random" trajectories it is possible to build a dataset $\mathcal{D} = \{(\mathbf{X}_i, \mathbf{Y}_i) | i = 1, ..., n_d\}$, where n_d is the number of sampled point. This dataset will be utilized by the two algorithms for the offline learning procedure.
It is important to enlight how the training trajectories have been chosen for this procedure.
The dataset is build on three different steps. Firstly the robot has been driven by an open-loop controller trying to track some random trajectories. However this was not sufficiently informative about the behaviour of the elastic term. In the following indeed, it will be illustrated how the non-linear MPC explores some "extreme states" (i.e. whenever the difference between $\boldsymbol{\theta}$ and $\mathbf{q}$ is very large) in order to compute the optimal torque command. Then it is necessary to teach the learning model how to predict the elastic term in a proper way even in these condition. Consequently a second dataset has been built with points collected by driving the robot through highly-exciting trajectories, in this way it is possible to predict correctly the elastic term in extreme situations. Eventually, by augmenting the first dataset with the second one, it is possible to train the model to predict the correct value of the elastic term in every situation. Aggiungere terzo step offline: Stimare da zero l'elasticità nonlinear e terzo step online: Partire da una stima iniziale (train su un dataset iniziale) ed andare a correggere il modello.

4.1 Gaussian Process Regression

Gaussian Process is a powerful machine learning technique. An application consists in estimating the relationships between a dependent variable and one or more independent variables, which, in statistical modeling, is also called regression. In the robotics field one of the main application of machine learning consists in using a large set of measurements and an appropriate algorithm, in order to improve the robot's performances in executing a given task, from the point of view of accuracy and repeatability.

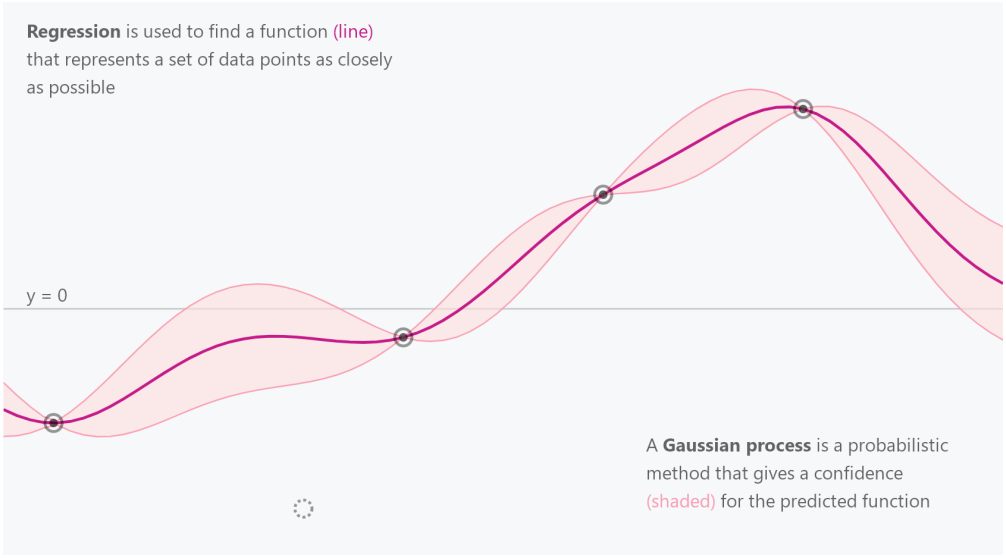


Figure 3: GP Visualization

Before going into the process details it is necessary to define the Gaussian probability distribution and recall some basic concepts in probability. A probability distribution is a description of how likely a random variable or set of random variables is to take on each of its possible states. However, rather than give the probability of a specific state directly, a probability density function (PDF) $p(x)$ is often used. The probability density function gives the probability of landing inside an infinitesimal region with volume δx and this probability is given by $p(x)\delta x$. A Gaussian probability distribution is defined by the mean vector μ which describes the expected value of the distribution and the variance which indicates how the actual realizations are spread out from the mean (covariance matrix Σ, in case of multivariate Gaussian distribution, which describes how the corresponding random variables are correlated). With correlation we mean the statistical measure that expresses the extent to which two variables are linearly related (meaning they change together at a constant rate). Correlation it's a common tool for describing simple relationships without making a statement about cause and effect. By writing $y \sim \mathcal{N}(\mu, \Sigma)$ we say y follows a normal distribution also called Gaussian distribution.
Gaussian distributions are closed under conditioning and marginalization, therefore the resulting distributions after these two operations are also Gaussian. Sometimes we know the probability distribution over a set of variables and we want to know the probability distribution over just a subset of them. The probability distribution over the subset is known as the marginal probability distribution. The operation which allows us to obtain this probability distribution is called Marginalization of a random variable, and consists in finding the probability density of a random variable given the joint probability density of two or more random variables.

$$p_X(x) = \int_y p_{X,Y}(x, y)dy = \int_y p_{X|Y}(x|y)p_Y(y)dy \tag{32}$$

In many cases, we are interested in the probability of some event, given that some other event has happened. This is called a conditional probability. We denote the conditional probability that $y = y*$ given $x = x*$ as $P_{Y|X}$.

$$X|Y \sim N(\mu_X + \Sigma_{XY}\Sigma_{YY}^{-1}(Y - \mu_Y),\ \Sigma_{XX} - \Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{YX}) \tag{33}$$

$$Y|X \sim \mathcal{N}(\mu_Y + \Sigma_{YX}\Sigma_{XX}^{-1}(X - \mu_X),\ \Sigma_{YY} - \Sigma_{YX}\Sigma_{YY}^{-1}\Sigma_{YX}) \tag{34}$$

Marginalization of a Gaussian PDF can be seen as integrating along one of the dimension of the Gaussian distribution while we can think about Conditioning as making a cut through the multivariate distribution, yielding a new Gaussian distribution with fewer dimensions.
The key idea is that Gaussian Process is a stochastic process, such that every finite collection X,Y has a multivariate normal distribution. In particular, since the information about the training set is available we are interested in the conditional probability $P_{Y|X}$ or in other words we want to condition the output of the process, given a certain input, with respect to the data set.
Now we have to understand how to compute the mean and the covariance of this distribution.
In order to build the covariance we use a covariance function which is often called the kernel of the Gaussian process.

$$k : \mathbb{R}^n \times \mathbb{R}^n \longmapsto \mathbb{R} \tag{35}$$

Where *n* is the dimension of the points. This function is evaluated for each pairwise combination of the given points. As we have mentioned before the *ij*-entry of this matrix define the correlation between the *i*-point and the *j*-point of the input, and how much influence they have on each other. For what concerns the mean, we can always assume that it is equal to zero. As a matter of fact, even if it was different from zero we can add it to the resulting function values after the prediction step to translate it. Now we are ready to compute the probability distribution, let's consider the joint distribution P_{X,X^*} between the test points *X** and the training points *X*, which is a multivariate Gaussian distribution and has dimension *dim(X)+dim(X*)*. The training set is divided into input x and output y, while with $f(x^*)$ we denote the function value at the point x^* belonging to the test set. Using conditioning we can obtain $P_{X^*|X}$ from P_{X,X^*} whose dimension is equal to the number of test points. The intuition behind this step is that the training points constraint the set of functions to those that pass through the training points.

$$P_{X,X^*} = \begin{bmatrix} y \\ f(x^*) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) & K(X,X^*) \\ K(X^*,X) & k(X^*,X^*) \end{bmatrix}\right) \tag{36}$$

Where with $K(A,B)$ we denote the covariance matrix between A and B built using the kernel function defined in equation eq. (35).
The predicted mean and the covariance for a single point are respectively given by:

$$\begin{aligned} f(x^*) &= k^{*T}(K)^{-1}y \\ V(x^*) &= k(x^*,x^*) - k^{*T}(K)^{-1}k^* \end{aligned} \tag{37}$$

4.2 Neural Network

A different approach consist in building an artificial neural network. This structure, in our case of shallow neural network, consists in three layers of neurons, where each neuron is linked to every neuron of the next layer, through an edge. Neurons and edges have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Then the learning process consists in finding the optimal values of these weights, in such a way that given a new input, the neural network is able to produce an output as close as possible to its real value. fig. 4 helps to visualize the structure of a neural network with three inputs, two output and a hidden layer.

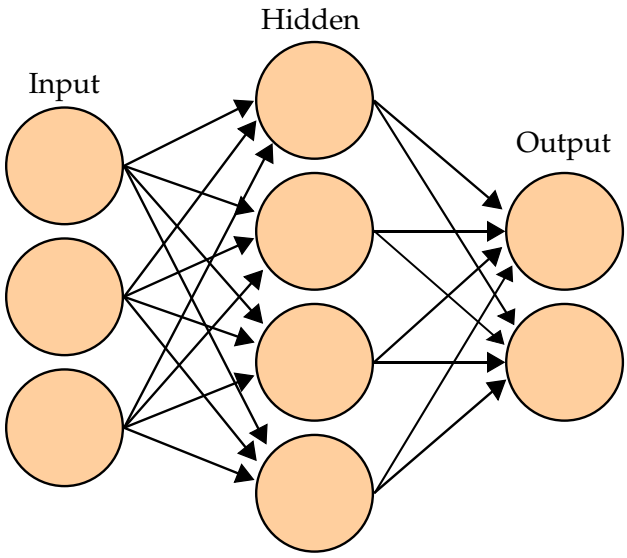


Figure 4: Neural Network Visualization

4.3 Comparison: GP vs NN

Since we want to employ the learning technique starting from a dataset with few datapoints, and then refine the learning online, we compared the performances of the Gaussian Process regressor and Neural Networks when trained on small initial datasets.

Below you can find the performances in terms of prediction error of the elastic terms for the two models, computed along *exciting trajectories* of the system, i.e. trajectories attained when supplying a torque with prominent oscillatory behavior.

In fig. 5 you can see the results of the test, performed for the Neural Network model. You can see both the evolution of the prediction error (for both components of the elastic term) throughout the test trajectory, and also the corresponding overall RMSE.

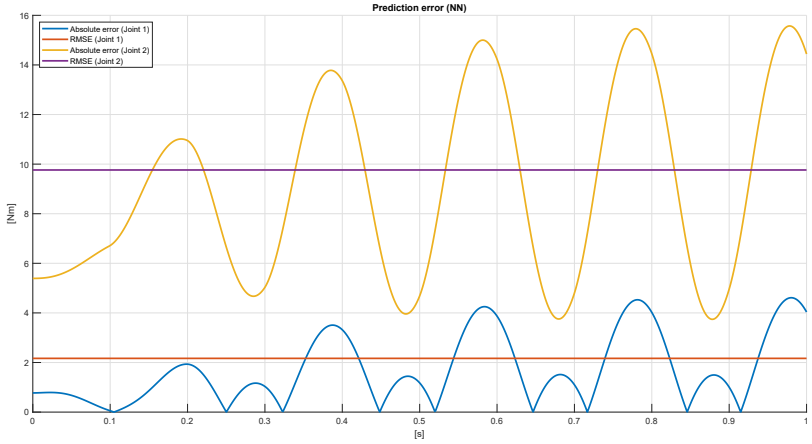


Figure 5: Prediction error of the Neural Network model.

In fig. 6 you can see the results of the same test, performed for the Gaussian Process regressor model.

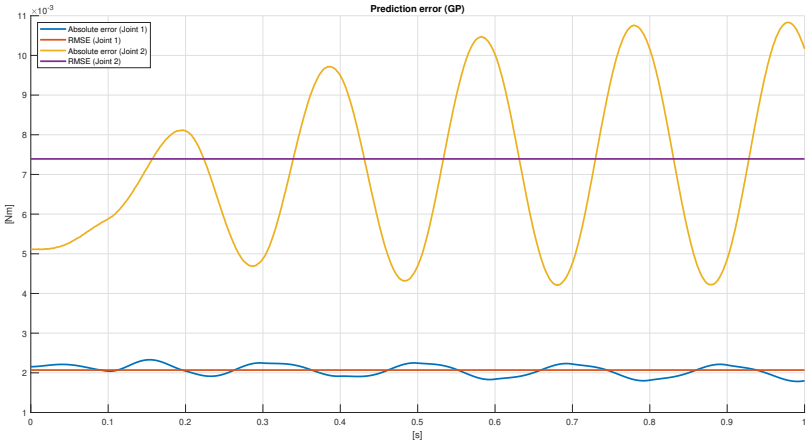


Figure 6: Prediction error of the Gaussian Process regressor model.

From the comparison above it is clear how the Gaussian Process regressor outperforms the Neural Network model on the task under consideration. Clearly, if we were to increase the size of the dataset, the NN model would close the gap with the GP model, but this is empirical confirmation that the GP regressor needs fewer datapoints with respect to the Neural Network to be trained effectively, and thus it is more suited to online learning.

5 Simulation setting

In the following we will describe the setting in which the simulations of our approach have been tested.

All the code has been written in MATLAB, leveraging its Model Predictive Control Toolbox for the nonlinear MPC and the Statistics and Machine Learning Toolbox for the implementation, training and prediction of the Gaussian Process Regressor.

The robot model as described in section 2 has been computed following a Lagrangian approach, i.e. defining symbolically the kinetic and potential energy of the system and then applying the Euler-Lagrange equations to derive symbolically the different terms of the model. The contributions from links and motors have been considered separately. The links and motors kinetic energy has been derived exploiting the *moving frames* algorithm, augmenting the links' masses with the masses of the corresponding motors. At the same time, given the assumptions underlying the model, the gravitational potential energy of both links and masses has been computed altogether.

Finally, for a complete extension to flexible joints, the definition of the motor inertia matrix $\mathbf{B}$ and of a proper nonlinear the elastic term is required. As a proof of concept, in our simulations we chose as nonlinear elastic term a simple third degree polynomial:

$$\boldsymbol{\psi}(\mathbf{q}-\boldsymbol{\theta}) = \mathbf{K}_1(\mathbf{q}-\boldsymbol{\theta}) + \mathbf{K}_2(\mathbf{q}-\boldsymbol{\theta})^3. \tag{38}$$

Although we have focused on a simple $2R$ planar robot under gravity, the procedure outlined above is very general and a model for a nR robot can be derived in the same manner, simply specifying the proper parameters with no change to the rest of the algorithm. The set of kinematics and dynamic parameters from which we derived the model can be found in table 1.

Robot parameters		
Mass	m	$1\ [kg]$
Link length	l	$1\ [m]$
CoM distance	d	$0.4\ [m]$
Link inertia matrix	$\mathbf{I}$	$\mathbf{I}_{3\times3}\ [kg\ m^2]$
Motor inertia matrix	$\mathbf{B}$	$\mathbf{I}_{3\times3}\ [kg\ m^2]$
Linear elastic coefficient	K_1	$10^3\ [Nm/rad]$
Nonlinear elastic coefficient	K_2	$10^2\ [Nm/(rad)^3]$
Damping coefficient	D	$10\ [Nm/(rad/s)]$

Table 1: Table of the kinematic and dynamic parameters of the robot model we have employed in our simulations. Whenever unspecified, the same parameter refers to both links.

Since MPC is naturally a discrete time control scheme, we also implemented a discretized version of the system dynamics with a simple Euler integration of the continuous dynamics. Formally, we compute

$$\mathbf{x}_{k+1} = \mathbf{x}_k + T_s\mathbf{f}(\mathbf{x},\mathbf{u}), \tag{39}$$

where $T_s = t_{k+1} - t_k$ is the *integration step* and is one of the control and simulation parameters (see table 2), and $f(x,u)$ is the vector field of the system dynamics expressed in state-space format:

$$\dot{\mathbf{x}} = \begin{pmatrix}\dot{\mathbf{q}}^T & \dot{\boldsymbol{\theta}}^T & \ddot{\mathbf{q}}^T & \ddot{\boldsymbol{\theta}}^T\end{pmatrix}^T = \mathbf{f}(\mathbf{x},\mathbf{u}). \tag{40}$$

As mentioned in the introductory paragraph, the task is of *setpoint regulation*, and in particular we specify a desired configuration for the $\mathbf{q}$ variables (that in principle might be computed through inverse kinematics to regulate the robot end-effector to some desired position in the workspace) and then simulate the closed-loop system (robot dynamics + Nonlinear MPC) evolution. Actually, as explained in section 3, the reference output variables are the motor variables $\boldsymbol{\theta}$. Therefore, since the robot is subject to gravity, to enforce stability of the desired forced equilibrium, we need to regulate $\boldsymbol{\theta}$ to a desired $\boldsymbol{\theta}_d$ such that the displacement between $\boldsymbol{\theta}$ and $\mathbf{q}$ at the equilibrium will give raise to an elastic force that compensates for gravity, so that the link variables $\mathbf{q}$ will be correctly maintained at the desired configuration $\mathbf{q}_d$. In particular, we can derive analytically the needed motor position $\boldsymbol{\theta}_d$ from the dynamics eq. (12) at the equilibrium $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\theta}, \dot{\boldsymbol{\theta}}, \ddot{\boldsymbol{\theta}}) = (\mathbf{q}_d, \mathbf{0}, \mathbf{0}, \boldsymbol{\theta}_d, \mathbf{0}, \mathbf{0})$:

$$\mathbf{g}(\mathbf{q}_d) + \boldsymbol{\psi}(\mathbf{q}_d - \boldsymbol{\theta}_d) = 0 \tag{41}$$

$$\mathbf{g}(\mathbf{q}_d) + \mathbf{K}_1(\mathbf{q}_d - \boldsymbol{\theta}_d) + \mathbf{K}_2(\mathbf{q}_d - \boldsymbol{\theta}_d)^3 = 0 \tag{42}$$

and equation eq. (42) is solved analytically for $\boldsymbol{\theta}_d$, since all the other quantities are known.

Last but not least, before commencing the simulation the robot model is linearized locally at the desired configuration $(\mathbf{q}_d, \boldsymbol{\theta}_d)$, in order to derive the needed LQR term for the terminal constraint, as explained in section 3. After these preliminary steps, the simulation can be started.

Simulation parameters		
Time to perform task	T	$1\ [s]$
Integration step	T_s	$10^{-3}\ [s]$
Maximum torque	$\mathbf{u}_{max}$	$100\ [Nm]$
Initial state	$\mathbf{x}_0$	$(\mathbf{0}_2, \mathbf{0}_2, \mathbf{0}_2, \mathbf{0}_2)\ [rad, rad/s]$
MPC control horizon	p	30
MPC *last steps*	k	5
State LQR weight	Q	$\mathrm{diag}\{10, 10, 1, 1, 0, 0, 0, 0\}$
Control LQR weight	R	$\mathrm{diag}\{1, 1\}$
Mixed LQR weight	N	$\mathbf{0}_{8\times 2}$

Table 2: Table of the simulation and MPC parameters we have employed in our simulations.

6 Simulation results

To validate the proposed approach, we performed several simulations in the setting described in the previous section. In what follows, we examine and make comparisons among four particular scenarios.

In a first scenario, we assume that the robot model is fully known. Thus, this first set of simulations is performed by using the data reported in Table 1 and simulation data in Table 2.

In a second scenario, we assume that the robot model is known by the prediction model of the MPC, but with *wrong* elastic parameters. In particular, we set the parameters known to MPC as $\tilde{K}_1 = 1.2 \cdot 10^3, \tilde{K}_2 = 1.2 \cdot 10^2$, i.e. a 20% error with respect to the real parameters. The idea is to demonstrate how such errors affect the convergence of our MPC-based approach.

In the third scenario, we assume that the elastic term is unknown and estimate it offline using the GP approach described in section 4.1.

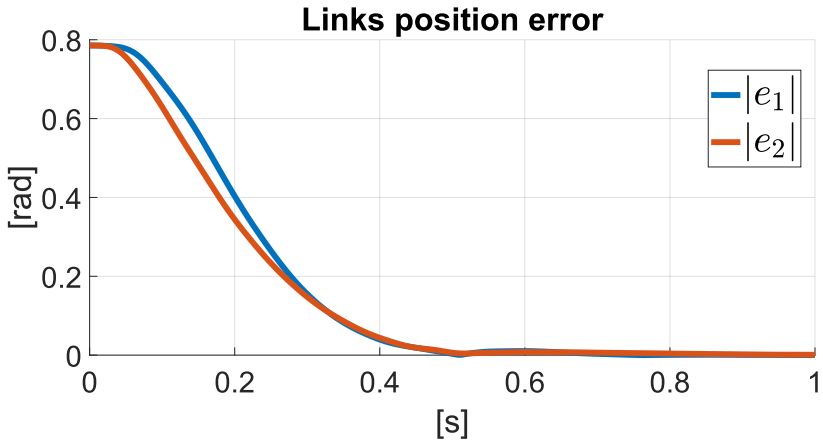
Finally, in the last set of simulations, learning is performed online starting from an estimation of the elastic term obtained by applying the GP approach offline on a reduced dataset.

To make consistent comparisons, we decided to report results obtained in the above scenarios simulating rest-to-rest trajectories resulting from the following two initial and final desired configurations:

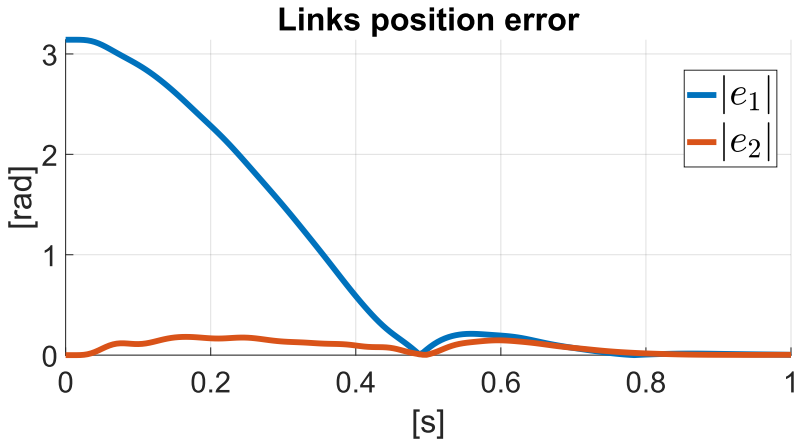
- (Case 1): $\mathbf{q}_{in} = \begin{pmatrix} 0 & 0 \end{pmatrix}^T \rightarrow \mathbf{q}_{fin} = \begin{pmatrix} \pi/4 & \pi/4 \end{pmatrix}^T$
- (Case 2): $\mathbf{q}_{in} = \begin{pmatrix} -\pi/2 & 0 \end{pmatrix}^T \rightarrow \mathbf{q}_{fin} = \begin{pmatrix} \pi/2 & 0 \end{pmatrix}$

6.1 Nominal MPC

In fig. 7 we can see the results of the simulations performed in the first of the scenarios outlined above: the model is completely known, and the focus is on establishing the behavior of MPC in *nominal* conditions, i.e. in the ideal situation in which the learning algorithm has converged perfectly to the unknown model.

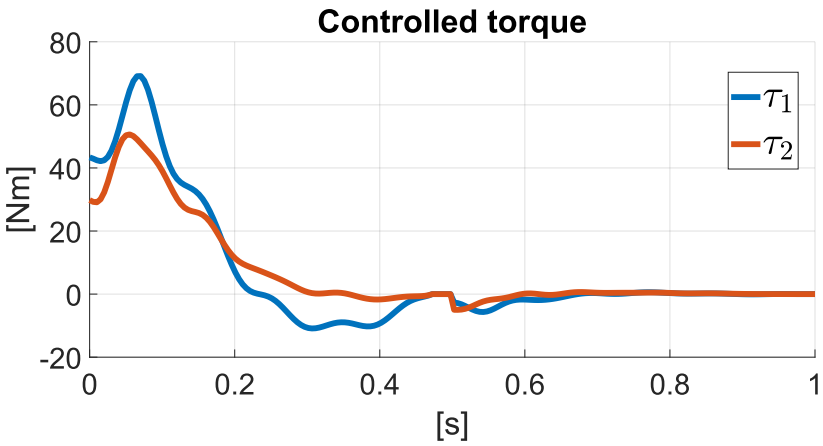


(a) Case 1

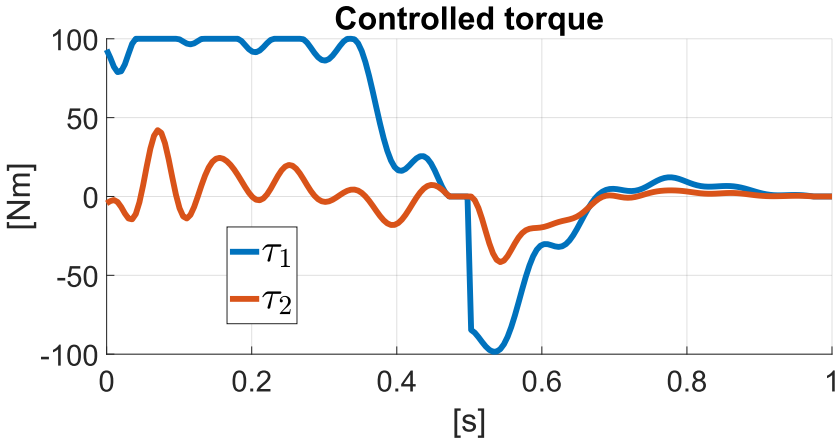


(b) Case 2

Figure 7: Links' position errors when driven by MPC in nominal conditions.



(a) Case 1



(b) Case 2

Figure 8: Torques issued by MPC in nominal conditions.

As we expect, since the elastic term is completely known, the nominal MPC performs very well, and this proves the efficiency of the MPC in nominal conditions. Notice also in fig. 8 how the controlled torque from MPC (without the gravity compensation) converges to zero when the desired equilibrium is reached. The goal is then to obtain similar performance in case the elastic term is predicted from the GP model.

6.2 Nominal MPC with wrong elasticity

In fig. 9 we can see the results of the simulations performed in the second of the scenarios outlined above: the model is completely known, but with the wrong elastic parameters.

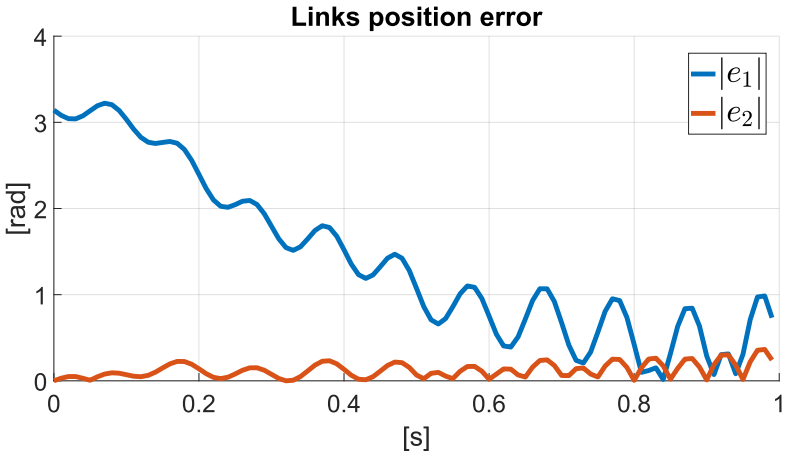


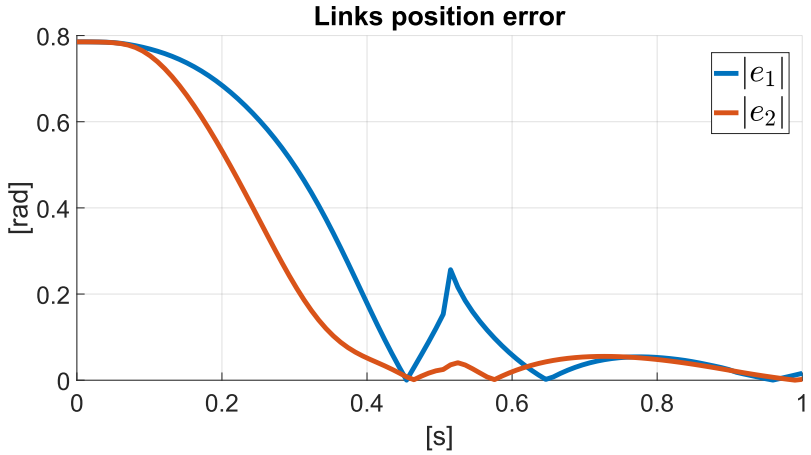
Figure 9: Links position errors with torques computed by MPC in nominal conditions but with wrong elasticity model.

We can see how the commands issued by MPC on the basis of a wrong prediction model, in particular one with a stiffer joint elasticity, result in the real model drifting a little too far at each timestep, and in a subsequent "too strong" correction at the next timestep. This correction is again based on the wrong model and thus introduces another even larger error, and these errors build up over time to produce the increasingly wide oscillations we observe.

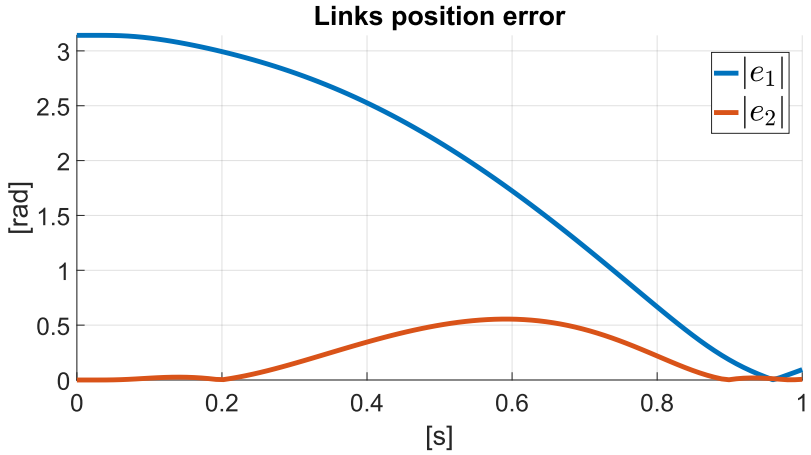
6.3 Gaussian Process with offline learning

In order to highlight the benefit of a well-suited dataset for the training of the model, in this section, the performance of the MPC joint with GP are compared upon two different datasets.

Figures fig. 10 show how performance is reduced by training on a reduced dataset. Such trend is more evident in the swing up case since it requires a better knowledge of the elastic term, due to the more exciting path that the robot needs to follow.



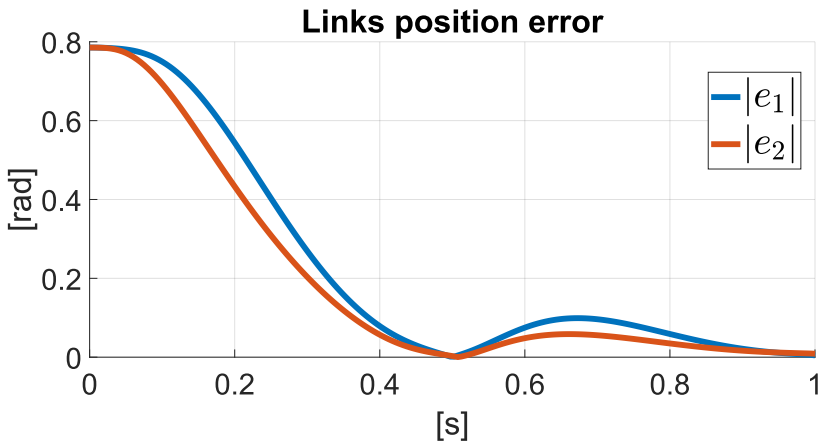
(a) Case 1



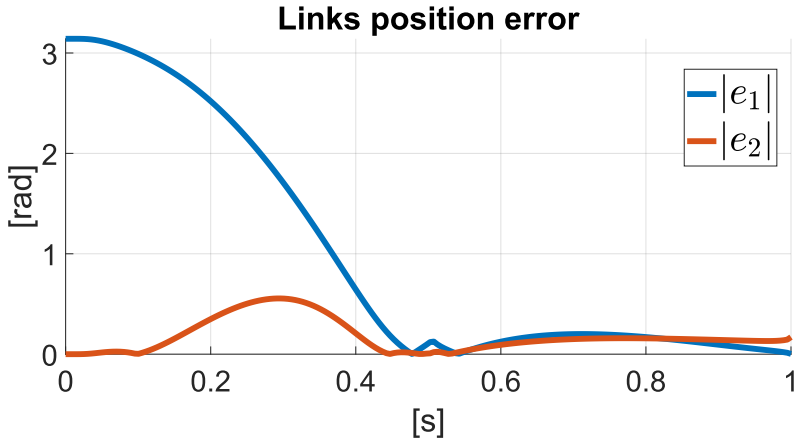
(b) Case 2

Figure 10: Links' position errors when driven with torques computed by MPC, combined with the GP regressor trained on a considerably small dataset.

In fig. 11 instead we can appreciated how the GP model, trained on a richer dataset (with 100 datapoints) which incorporates also what we call "*extreme configurations*", i.e. configurations that feature a large displacement between $\mathbf{q}$ and $\boldsymbol{\theta}$, yields very good results. Indeed, such graphs come substantially closer to fig. 7, which show results in nominal conditions.



(a) Case 1

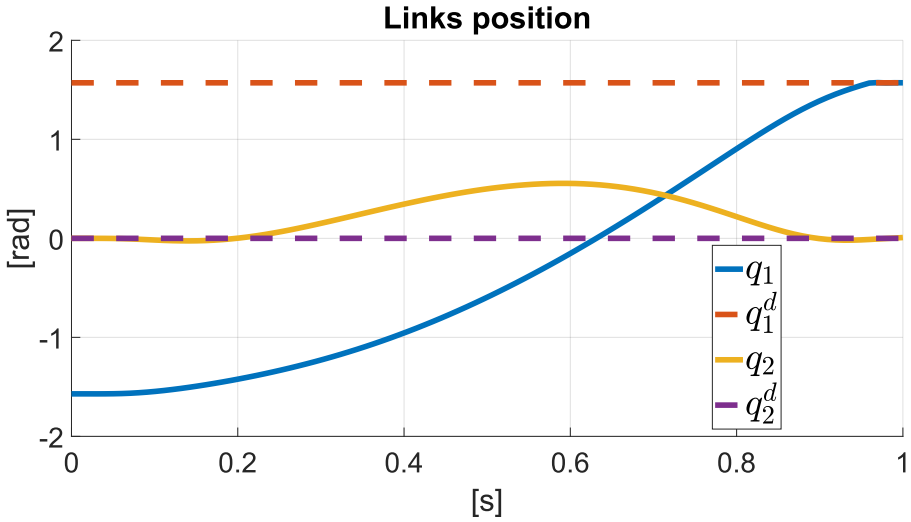


(b) Case 2

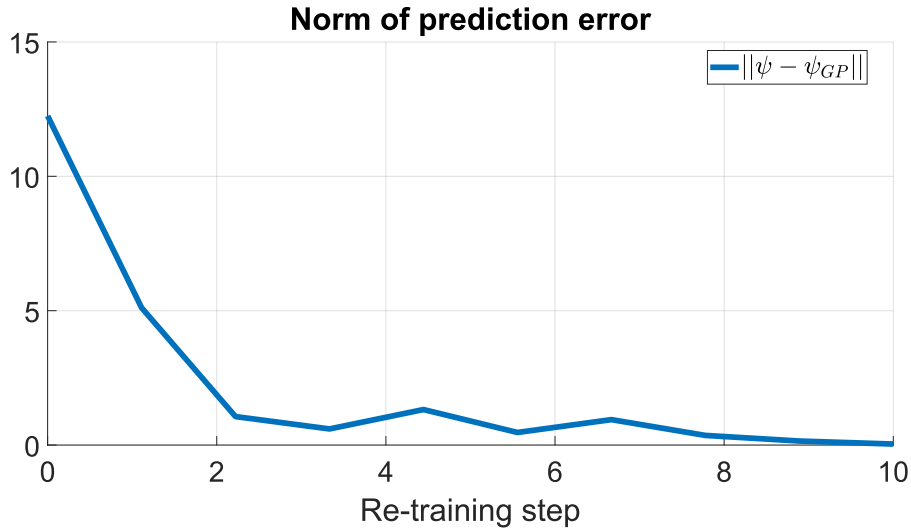
Figure 11: Links' position errors when driven with torques computed by MPC, combined with the GP regressor trained on a larger dataset obtained by sampling exciting trajectories.

6.4 Gaussian Process with online learning

A similar discussion can be done when the GP model is trained while the robot is moving. We refer to this procedure as *online learning*. In this case it can be appreciated how adding points to the dataset and performing again the training step, leads to better performance.



(a) Links' positions



(b) GP prediction error

Figure 12: Links' positions and GP prediction error when the GP model is refined online with 10 retraining steps, increasing the size of the dataset by 10 datapoints each time.

As you can see in fig. 12b the prediction error, although with some fluctuation, steadily decreases until reaching a value of 0.0667 for the error norm. This leads to ability for MPC of more precisely simulating the evolution of the system, thue issuing more precise commands that lead to links' positions converge to the reference positions with increased precision with respect to the case with no online learning, as can be seen in fig. 12a.

7 Conclusions

We would like to conclude this report by underlining some possibilities for future work on the project.

First of all, it would be interesting to study the behavior of the presented approach when dealing with the problem of trajectory tracking, instead of the somewhat simpler problem of regulation. Although we believe this extension to be straightforward, it has not been considered in the project due to time and computational resources constraints.

Then, we have considered a number of possibilities to directly address the problem of instability in MPC (the so-called "extreme configurations"). In particular, two approaches have been outlined, but not implemented: the first one would try to avoid exploring regions of the state space corresponding to high uncertainty of the learning model, by the addition of a variance-dependent term to the cost function; on the other hand, the second one would try to avoid exploring regions of the state space corresponding to large displacements between links and motors positions (and thus eliminating at the root the problem of "extreme configurations"). We suppose that both approach should lead to more stable behavior since the predicted elastic term within the prediction model of MPC should match more closely the ground truth one.

Finally, we considered replacing the deterministic constraints with the so-called *chance constraints* that are instead probabilistic constraints. This would imply the use of *Stochastic MPC* that should result in a more robust behavior in the presence of model uncertainties.

References

[1] 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 *Conference on Robot Learning*, pp. 1359–1368, PMLR, 2020.

[2] M. Capotondi, G. Turrisi, C. Gaz, V. Modugno, G. Oriolo, and A. De Luca, "Learning feedback linearization control without torque measurements," IRIM, 2020.

[3] B. Siciliano and O. Khatib, *Springer handbook of robotics*. springer, 2016.

[4] M. W. Spong, "Modeling and control of elastic joint robots," *Journal of Dynamic Systems, Measurement, and Control*, 1987.

[5] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010.

[6] G. Oriolo and Y. Nakamura, "Control of mechanical systems with second-order nonholonomic constraints: Underactuated manipulators," in *Conference on Decision and Control*, pp. 2398–2403, 1991.

[7] A. De Luca, S. Iannitti, R. Mattone, and G. Oriolo, "Control problems in underactuated manipulators," in *2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics. Proceedings (Cat. No. 01TH8556)*, vol. 2, pp. 855–861, IEEE, 2001.

[8] I. Fantoni, R. Lozano, and S. Sinha, "Non-linear control for underactuated mechanical systems," *Appl. Mech. Rev.*, vol. 55, no. 4, pp. B67–B68, 2002.

[9] A. Isidori, *Nonlinear control systems*. Springer Science & Business Media, 2013.

[10] A. Bemporad, "Model predictive control." Course slides, 2020.

[11] D. Mayne, "An apologia for stabilising terminal conditions in model predictive control," *International Journal of Control*, vol. 86, 11 2013.

[12] A. D. Luca, "A review on the control of flexible joint manipulators." Workshop on Soft Robotic Modeling and Control, 2018.

[13] P. Tomei, "A simple pd controller for robots with elastic joints," *IEEE Transactions on automatic control*, vol. 36, no. 10, pp. 1208–1213, 1991.