# Robotics II Project Report

Fioretti Gianmarco 1762135,
Mauro Alessandro 1755066,
Nicotra Emanuele 1757179

October, 2020

## Contents

# 1 Introduction

In order to make a robot system execute tasks as regulation and trajectory tracking it needs definitely a feedback control which effectively correct the errors of the feed-forward commands. Standard control methods consist in the inversion of the dynamics model in order to calculate the necessary torque that minimizes the error between the robot reference configuration and the actual one. A way more precise, energy-efficient and suitable for complaint robots control method is the so-called Computed Torque which corresponds to use the dynamic model over the system considering the joint weakly coupled. However, this method requires to exactly know the dynamic robot model and therefore it is not an optimal neither a robust control method. Therefore almost always we have some mismatch between the dynamic model (the nominal one) and the actual behavior of the robot, also after identification procedures. In such cases it is convenient to implement robot learning algorithms in order to better approximate the dynamic model and being able apply a correction to the Feedback Linearization (FL) and so correctly linearize the system. In particular, in this project, we are going to implement the Gaussian Process Regression over the corresponding torque errors between the nominal dynamic models and the "real one".
To perform tests and simulations we will use the Peter Cork's Robotics Matlab tool.

## 1.1 Robot Models

In this project we are considering a 2R planar Robot. The real robot differs from the nominal model by some mismatches in masses and DH parameters, as follows:

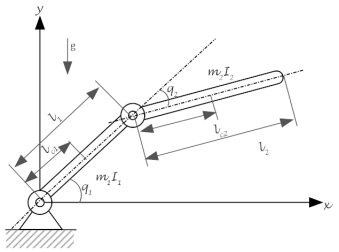


Figure 1: *2R Scheme*

| | Link1 | Link2 |
|---|---|---|
| $a$ | 2 | 1 |
| $\theta$ | $q_1$ | $q_2$ |
| $d$ | 0 | 0 |
| $\alpha$ | 0 | 0 |
| $m$ | 1 | 1 |

Table 1: Nominal Robot Model

| | Link1 | Link2 |
|---|---|---|
| $a$ | 2.1 | 0.8 |
| $\theta$ | $q_1$ | $q_2$ |
| $d$ | 0 | 0 |
| $\alpha$ | 0 | 0 |
| $m$ | 1 | 1.16 |

Table 2: Real Robot Model

## 1.2 Computed Torque

The robot dynamics is expressed using the Canonical Lagrangian Form:

$$M(q)\ddot{q} + c(q,\dot{q}) + g(q) = u \tag{1}$$

Where $M \in \mathbb{R}^{2\times 2}$ is the dynamic matrix, $c \in \mathbb{R}^2$ is the Coriolis and centrifugal term and $g \in \mathbb{R}^2$ is the gravity vector, while $q \in \mathbb{R}^2$ represents the configuration at time $t$. Assuming no model mismatch between the real robot model and the nominal one and using a PD controller with gains $K_p$ and $K_d$ which are diagonal $2 \times 2$ matrices. The controller defined as the cascade of FL and the kinematic PD is:

$$u = M(q)(\ddot{q}_d + K_p(q_d - q) + K_d(\dot{q}_d - \dot{q})) + c(q,\dot{q})\dot{q} + g \tag{2}$$

The goal of this kind of control is to eliminate the nonlinearities in the dynamics as shown in Figure 2. Combining (1) and (2) we end up to an error equation for the closed-loop equal to:

$$\ddot{e} + K_p e + K_d \dot{e} = 0, \text{ with } e = q_d - q \tag{3}$$

so that the feedback gains $K_p$ and $K_d$ can be chosen so that (3) is stable. Hence, in case of using regression techniques for model approximation the equation (2) can be simply written as:

$$u = f(q_d, \dot{q}_d, \ddot{q}_d + K_p e + K_d \dot{e}) \tag{4}$$

where $f$ represents the approximated model through a certain robot learning method. In our specific case, $f$ will correspond to the nominal dynamic model computed in $(q_d, \dot{q}_d, \ddot{q}_d + K_p e + K_d \dot{e})$ plus the torque difference predicted by the Gaussian Process which has to appropriately correct the given mismatches.

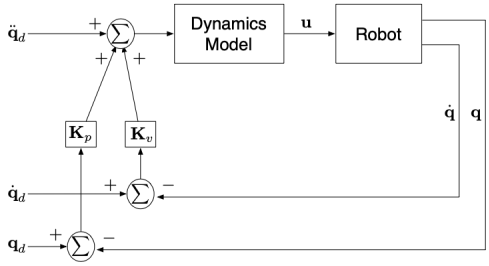


Figure 2: *Inverse Dynamics Control* [**1**]

Using the parameters in the following table we provide a simulation of Feedback Linearization control law applied to the 2R manipulator.

| | Joint1 | Joint2 |
|---|---|---|
| $q_0$ | 0 | $-\frac{\pi}{2}$ |
| $\dot{q}_0$ | 0 | 0 |
| $q_f$ | $4\pi$ | $\frac{\pi}{2}$ |
| $K_p$ | 1000 | 1000 |
| $K_d$ | 100 | 100 |

Table 3: Simulation parameters.

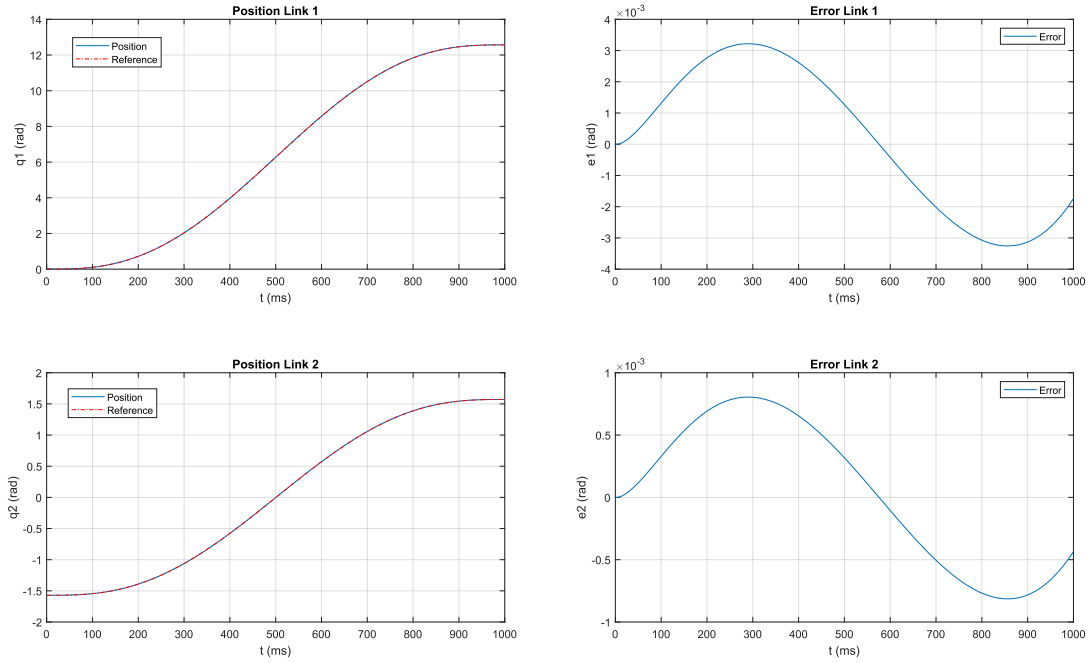


Figure 3: Plot of Position and Error.

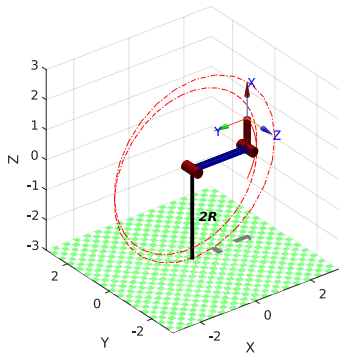


Figure 4: Plot 3D

By a naive look one might thinks that the robot is performing the task perfectly, but actually there is no mismatch yet.

# 2 Theoretical Explanation

## 2.1 Gaussian Process

Gaussian Process is one of the most used 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. 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 $\delta x$ and this probability is given by $p(x)\delta x$. A Gaussian probability distribution is defined by the mean vector $\mu$ 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 $\Sigma$, 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.

$$p_X(x) = \int_y p_{X,Y}(x, y)dy = \int_y p_{X|Y}(x|y)p_Y(y)dy \tag{5}$$

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 $P_{Y|X}$.

$$X|Y \sim N(\mu_X + \Sigma_{XY}\Sigma_{YY}^{-1}(Y - \mu_Y),\ \Sigma_{XX} - \Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{YX}) \tag{6}$$

$$Y|X \sim \mathcal{N}(\mu_Y + \Sigma_{YX}\Sigma_{XX}^{-1}(X - \mu_X),\ \Sigma_{YY} - \Sigma_{YX}\Sigma_{YY}^{-1}\Sigma_{YX}) \tag{7}$$

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.

$$k : \mathbb{R}^n \times \mathbb{R}^n \longmapsto \mathbb{R} \tag{8}$$

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 $P_{\mathrm{X^*|X}}$ from $P_{\mathrm{X,X^*}}$ 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.

$$P_{X,X^*} = \begin{bmatrix} y \\ f(x^*) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) & K(X,X^*) \\ K(X^*,X) & k(X^*,X^*) \end{bmatrix}\right) \tag{9}$$

Where with $K(A,B)$ we denote the covariance matrix between A and B built using the kernel function defined in equation (8).
The last step is to consider that in a real-word scenario the training points are not perfect measurements but there is always a measurement error due to some uncertainty. The solution to this problem consists in adding a noise term to the training points.

$$y = f(x) + \epsilon \tag{10}$$

where the error is also given by a normal distribution

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2) \tag{11}$$

In this case the joint distribution becomes:

$$P_{X,X^*} = \begin{bmatrix} y \\ f(x^*) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X,X^*) \\ K(X^*,X) & k(X^*,X^*) \end{bmatrix}\right) \tag{12}$$

And the predicted mean and the covariance for a single point are respectively given by:

$$\begin{aligned} f(x^*) &= k^{*T}(K + \sigma_n^2 I)^{-1} y \\ V(x^*) &= k(x^*,x^*) - k^{*T}(K + \sigma_n^2 I)^{-1} k^* \end{aligned} \tag{13}$$

where $K = K(X,X)$ is the covariance matrix computed between the training points, $k^{*T} = K(X^*,X)$ is the covariance between the points to be predicted and the training points.

## 2.2 Sparsification

Sparse Gaussian Process allows to perform regression on very large data sets. There are many ways to sparsify a Gaussian Process, i.e. to avoid using all the available points in the data set since its complexity scales to $\mathcal{O}(n^3)$, such as kernel approximations, local versions of GPR and mostly the identification of a representative subset of data points, taking advantage of the latter approach the simplest possible solution is to discard the other remaining data points and to perform a standard GPR over the selected ones only. Instead, here we have used the *Projected Process Approximation* to induce the sparse model approximation thanks to the Nystrøm method. Such an approximation consists in estimating the likelihood of the entire set of data (of size $n$) through the projection of the selected subset points (of size $m$) to the $n$ dimensions, avoiding in fact the computational load which otherwise would make it impractical. In a nutshell, we are trading off a little accuracy decrease with a huge of computation efficiency. Thanks to which we are now able to perform both offline and online regressions.

First of all, we need to have the representative and meaningful subset of data, from now on let's call it *active subset*, through which the model can regress. A variety of selection and deletion criteria have been proposed over the years, in this project we have used the greedy *maximum error* one introduced by [**3**] due to its space and time complexity $O(1)$ per remaining point, same as for random selection, making real time problems possible. The size of the active subset $m$ depends on the available machine hardware, so it can be considered fixed and well-known upon formulation.

Once have selected the representative subset of data, we can train the Sparse Gaussian Process model by first estimating the marginal log likelihood and then by optimizing the hyperparameters $\boldsymbol{\theta}$ in order to maximize

it. For the offline regression the process of subset selection and parameters optimization is accomplished only once, whereas for the online version (*online learning*) it is fulfilled continuously as new input points arrive. In such an approach the simulations of trajectories never seen before by the manipulator and the optimization stage therefore got together.

### 2.2.1 Sparse GPR

Given a data set as follows $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_n, \mathbf{y}_n)\}$ where $\mathbf{x}_i$ is the joint position-velocity-acceleration vector and $\mathbf{y}_i = \mathbf{f}_i + \epsilon$ is the noisy realizations of the corresponding torque error due to the mismatches between the real and the nominal robot dynamic models, with $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. Let $\mathbf{X}$ be the overall matrix composed by all the $\mathbf{x}_i$ vectors as columns, and let $I \subset \{1, \cdots, n\}$ be the active set of size $m$ namely the set of indexes which indicate the active subset of points and let $R = \{1, \cdots, n\} \backslash I$ be the indexes of the remaining points such that $I \cup R$ represent all the input training points of the data set.
Given the aforementioned data set subdivision in active and remaining points and the row rearrangement such that the former are placed first than the others, the whole covariance matrix is composed as follows

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_I & \mathbf{K}_{I,\cdot} \\ \mathbf{K}_{I,\cdot}^T & \mathbf{K}_R \end{bmatrix}_{n \times n} \tag{14}$$

where $\mathbf{K}_I \in \mathbb{R}^{m \times m}$ is the covariance matrix between the points belonging to the active subset while $\mathbf{K}_{I,\cdot} \in \mathbb{R}^{m \times n}$ is the one between them and all the training points.
The main goal is to estimate the conditional distribution $P(\mathbf{f}_I | \mathbf{y}, \mathbf{X})$ in order to compute the mean and variance of unseen data points. According to Bayes Theorem it holds true that

$$P(\mathbf{f}_I | \mathbf{y}, \mathbf{X}) \propto P(\mathbf{y} | \mathbf{f}_I, \mathbf{X}) P(\mathbf{f}_I | \mathbf{X}) \tag{15}$$

We can therefore introduce the following approximation of the element $P(\mathbf{y} | \mathbf{f}_I, \mathbf{X})$ through which the information of all the data points are encoded into the active subset in a sense

$$Q(\mathbf{y} | \mathbf{f}_I, \mathbf{X}) = \mathcal{N}(\mathbf{y} | \mathbf{P}_I^T \mathbf{f}_I; \sigma_n^2 \mathbf{I}) \tag{16}$$

where $\mathbf{f}_I$ is the actual outputs of the active subset and $\mathbf{P}_I = \mathbf{K}_I^{-1} \mathbf{K}_{I,\cdot} \in \mathbb{R}^{m \times n}$ is the projection matrix which maps $\mathbf{f}_I$ to the prior conditional mean $E[\mathbf{f} | \mathbf{f}_I]$. On the other hand

$$P(\mathbf{f}_I | \mathbf{X}) = \mathcal{N}(\mathbf{f}_I | \mathbf{0}, \mathbf{K}_I) \tag{17}$$

so that we can end up with the following estimated posterior distribution

$$Q(\mathbf{f}_I | \mathbf{y}, \mathbf{X}) = \mathcal{N}(\mathbf{f}_I | \mathbf{K}_I (\mathbf{K}_I \sigma_n^2 + \mathbf{K}_{I,\cdot} \mathbf{K}_{I,\cdot}^T)^{-1} \mathbf{K}_{I,\cdot} \mathbf{y}; \sigma_n^2 \mathbf{K}_I^{-1} (\sigma_n^2 \mathbf{K}_I + \mathbf{K}_{I,\cdot} \mathbf{K}_{I,\cdot}^T) \mathbf{K}_I^{-1}) \tag{18}$$

which, thanks to the Nystrøm factor and the Cholesky decomposition, can be also written as

$$Q(\mathbf{f}_I | \mathbf{y}, \mathbf{X}) = \mathcal{N}(\mathbf{f}_I | \mathbf{L} \mathbf{M}^{-1} \mathbf{V} \mathbf{y}; \sigma_n^2 \mathbf{L} \mathbf{M}^{-1} \mathbf{L}^T) \tag{19}$$

with $\mathbf{L} \in \mathbb{R}^{m \times m}$ is the lower Cholesky factor of $\mathbf{K}_I \in \mathbb{R}^{m \times m}$ used to efficiently perform matrix inversion, $\mathbf{V} = \mathbf{L}^{-1} \mathbf{K}_{I,\cdot} \in \mathbb{R}^{m \times n}$ is the Nystrøm factor for approximating the full covariance matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$, $\mathbf{M} = \mathbf{V} \mathbf{V}^T + \sigma_n^2 \mathbf{I} \in \mathbb{R}^{m \times m}$. Indeed integrating over the product $Q(\mathbf{f}_I | \mathbf{y}, \mathbf{X}) P(\mathbf{f}_I | \mathbf{X})$ the approximated marginal likelihood results in

$$Q(\mathbf{y} | \mathbf{X}) = \mathcal{N}(\mathbf{y} | \mathbf{0}; \mathbf{V}^T \mathbf{V} + \sigma_n^2 \mathbf{I}) \tag{20}$$

whose covariance is indeed approximated through Nystrøm as follows

$$\mathbf{V}^T \mathbf{V} = \mathbf{K}_{I,\cdot}^T \mathbf{L}^{-T} \mathbf{L}^{-1} \mathbf{K}_{I,\cdot} = \mathbf{K}_{I,\cdot}^T \mathbf{K}_I^{-1} \mathbf{K}_{I,\cdot} = \tilde{\mathbf{K}} \tag{21}$$

according to the inverse of the Cholesky decomposition

$$\mathbf{K}_I^{-1} = (\mathbf{L} \mathbf{L}^T)^{-1} = \mathbf{L}^{-T} \mathbf{L}^{-1} \tag{22}$$

At this point, as in equation (13) the predictive distribution for a test point $\mathbf{x}_*$ is given by

$$
\begin{aligned}
E[f(\mathbf{x}_*)] &= \mathbf{K}_{I,*}^T \mathbf{K}_I^{-1} E[Q(\mathbf{f}_I|y)] \\
&= \mathbf{K}_{I,*}^T (\mathbf{K}_I \sigma_n^2 + \mathbf{K}_{I,\cdot} \mathbf{K}_{I,\cdot}^T)^{-1} \mathbf{K}_{I,\cdot} \mathbf{y} \\
V(f(\mathbf{x}_*)) &= \mathbf{k}_{**} - \mathbf{K}_{I,*}^T \mathbf{K}_I^{-1} + \mathbf{K}_{I,*}^T \mathbf{K}_I^{-1} cov(Q(\mathbf{f}_I|y)) \mathbf{K}_I^{-1} \mathbf{K}_{I,*}^T \\
&= \mathbf{k}_{**} - \mathbf{K}_{I,*}^T \mathbf{K}_I^{-1} \mathbf{K}_{I,*} + \sigma_n^2 \mathbf{K}_{I,*}^T (\sigma_n^2 \mathbf{K}_I + \mathbf{K}_{I,\cdot} \mathbf{K}_{I,\cdot}^T)^{-1} \mathbf{K}_{I,*}
\end{aligned}
\tag{23}
$$

or equivalently

$$
Q(\mathbf{f}_*|\mathbf{x}_*, \mathbf{y}, \mathbf{X}) = \mathcal{N}(\mathbf{f}_*|\mathbf{K}_{I,*}^T \mathbf{L}^{-T} \mathbf{M}^{-1} \mathbf{V}\mathbf{y};\ \mathbf{k}_{**} - \|\mathbf{L}^{-1}\mathbf{k}_{I,*}\|^2 + \sigma_n^2 \|\mathbf{L}_M^{-1}\mathbf{L}^{-1}\mathbf{k}_{I,*}\|^2) \tag{24}
$$

where $\mathbf{K}_{I,*}$ is the covariance matrix between the active subset and the test point, $\mathbf{L}_\mathbf{M} \in \mathbb{R}^{m \times m}$ is the lower Cholesky factor of $\mathbf{M}$.
Hence, the approximated posterior distribution for all the training points induced through the active subset is given by

$$
Q(\mathbf{f}|\mathbf{y}, \mathbf{X}) = \mathcal{N}(\mathbf{f}|\tilde{\mathbf{K}}(\tilde{\mathbf{K}} + \sigma_n^2 \mathbf{I})^{-1} \tilde{\mathbf{K}}\mathbf{y}; \boldsymbol{\Sigma}) \tag{25}
$$

for some covariance $\boldsymbol{\Sigma}$, or equivalently

$$
Q(\mathbf{f}|\mathbf{y}, \mathbf{X}) = \mathcal{N}(\mathbf{f}|\mathbf{V}^T \mathbf{M}^{-1} \mathbf{V}\mathbf{y}; \boldsymbol{\Sigma}) \tag{26}
$$

whose mean vector $\boldsymbol{\mu}_I = E[Q(\mathbf{f}|\mathbf{y}, \mathbf{X})] \in \mathbb{R}^n$ will be exploited by the *maximum error* greedy criterion during the active subset selection (see next paragraph).

### 2.2.2 Heuristic

Many criteria have been proposed over the years, such as those which maximize the Information Gain, those which minimize the Kullback–Leibler divergence and so on, but since we are pursuing computation efficiency we opted for the greedy insertion and deletion criteria proposed by [**3**], used to determine the optimal subset of active points, because of the rapidity by which it can select a given remaining data point.
Let $I' = I \cup \{i\}$ be the index set of active points plus the $i$-th one currently belonging to $R$. In order to maximize the approximated posterior likelihood (26) the Euclidean norm $\|\mathbf{y} - \boldsymbol{\mu}_{I'}\|$ has to be minimized, where $\boldsymbol{\mu}_{I'}$ is equal to the estimated mean vector of the sparse model induced by the index set $I'$. This can be readily obtained by selecting the remaining point which has the *maximum error* with respect to the current posterior model. Therefore the **ME** insertion criterion is given by

$$
{}_{ME}\Delta_i = |\mathbf{y}_i - \boldsymbol{\mu}_{I,i}| \tag{27}
$$

Following the same reasoning a **ME** deletion criterion can be defined, in order to reduce redundancy in the greedily selected active subset or at least just slightly deteriorate the model accuracy in case a machine hardware requirement (about the size $m$) must be met, as follows

$$
{}_{ME}\nabla_i = |{}_{ME}\Delta_i \boldsymbol{\alpha}_{I,i}| \tag{28}
$$

where $\boldsymbol{\alpha}_I = \mathbf{R}^{-1}\mathbf{Q}^T \mathbf{K}_{I,\cdot}\mathbf{y} \in \mathbb{R}^m$ is this time computed through the QR decomposition for numerical stability ($\mathbf{QR} = \mathbf{LML}^T$). So that the active point with minimal value of the deletion criterion has to be removed from the active index set $I$.
The latter criterion turns out to be fundamental especially for the online learning into which new training points arrive continuously, where this heuristic process has to be therefore performed again and again.

# 3 Implementation

## 3.1 Data Generation

The first step of the GPR consists in building a sufficiently large data set. It is necessary in order to have the set of points needed to compute the training step and the inference test step of the algorithm. First, we compute a set of random trajectories using the quintic polynomial algorithm, which allows us to compute the desired trajectory taking as input the initial and final value of the position, velocity and acceleration. Then, being in a simulation, from the well-known dynamics models of the two robots, we can extract the values of the corresponding torques we are looking for, essentially performing the direct dynamics:

$$M(q)\ddot{q} + c(q, \dot{q}) + g(q) = u \tag{29}$$

## 3.2 GP Training

Given the set of $n$ training data points, which is divided in:

$$x_{\mathrm{i}} = [q_{\mathrm{i}}, \dot{q}_{\mathrm{i}}, \ddot{q}_{\mathrm{i}}] \tag{30}$$

$$y_{\mathrm{i}} = \Delta u_{\mathrm{i}} \tag{31}$$

We would like to learn a function $f(x_{\mathrm{i}})$ transforming the input vector into the target value $y_{\mathrm{i}}$.

$$y_{\mathrm{i}} = f(x_{\mathrm{i}}) + \epsilon \tag{32}$$

y is described by a Gaussian distribution

$$y \sim \mathcal{N}(0, K(X, X) + \sigma_n^2 I) \tag{33}$$

Where $X$ is the set containing all input points $x_{\mathrm{i}}$, $K(X,X)$ is the covariance matrix, while $\sigma_n$ is the noise variance.
In order to compute the covariance matrix, we use the Radial Basis Function kernel RBF:

$$k : \mathbb{R}^3 \times \mathbb{R}^3 \longmapsto \mathbb{R} \tag{34}$$

$$k(x, x') = \sigma e^{-\frac{1}{2}(x-x')^T)W(x-x')} \tag{35}$$

Where $W$ is a diagonal matrix, whose elements along the diagonal represent a sort of weight one for each dimension, more precisely, they are referred to as *characteristic length-scales*.

### 3.2.1 Hyperparameters Optimization

So far we have used the training set in order to get an approximation of $f(\cdot)$. This is achieved by optimizing values of the hyperparameters of the model, namely those used in the kernel function and the noise variance. For this task is necessary to consider a certain error, which is the difference between the predicted and the measured outputs. By minimizing this error we can adjust the hyperparameters of the model. $\sigma$ and $l$ are those parameters to be estimated involved in the RBF kernel function, so in conclusion we must adjust a total of four hyperparameters.
The hyperparameters' optimal values are estimated using a loss function, a function that maps an event or values of one or more variables onto a real number. Intuitively it represents some "cost" associated with the event.

Such a cost function is represented by the inverse of the marginal likelihood over the entire data set, which is called as such since it is obtained through marginalization over the latent functions $\mathbf{f}$. By definition, a marginal likelihood is proportional to the data-fit but at the same time penalize the model complexity, by fostering the least complex model which well fits the training data.

The Logarithmic Marginal Likelihood conditioned on the hyperparameters of a standard Gaussian Process model is defined as

$$log\ p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}) = \frac{1}{2}\mathbf{y}^T(K+\sigma_n^2 I)^{-1}\mathbf{y} + \frac{1}{2}log|K+\sigma_n^2 I| + \frac{n}{2}log(2\pi) \tag{36}$$

the $-\frac{1}{2}\mathbf{y}^T(K+\sigma_n^2 I)^{-1}\mathbf{y}$ term is the most intuitive one, namely the one which induces the model to actually explain the data, whereas the logarithmic determinant of the covariance matrix represents the complexity penalty, and finally the third one represents the normalization factor.

Again, according to the adopted sparse approach it can be simplified as follows

$$log\ p(\mathbf{y}|X,\boldsymbol{\theta}) = (n-m)log(\sigma_n) + \sum_{i=1}^{m} log(l_{M,ii}) + \frac{1}{2\sigma^2}(\mathbf{y}^T\mathbf{y} - \boldsymbol{\beta}_I^T\boldsymbol{\beta}_I) + \frac{n}{2}log(2\pi) + \frac{1}{2\sigma^2}trace(K - V^TV) \tag{37}$$

where, as above, $n$ stands for the total number of data points, $m$ is the size of the active subset, and $\boldsymbol{\beta}_I = L_M^{-1}Vy$.

Finally, we have implemented a simple method following the Expectation Maximization paradigm in order to further improve the hyperparameters optimization. Let such hyperparameters $\boldsymbol{\theta}$ be the latent values, initially guessed at random. Given that the active subset selection involves the computation of the estimated mean vector of the sparse model $\boldsymbol{\mu}_I$, which depends on the hyperparameters themselves, and that it is performed while those values are kept fixed perhaps the bad initial guess of $\boldsymbol{\theta}$ makes the model to poorly estimate the $\boldsymbol{\mu}_I$ and so to poorly select the active points. Thus, we iteratively perform the Expectation step by selecting the active subset through the current guess of the hyperparameters, and then the Maximization step by optimizing them through any gradient based method. Repeating this process our model will converge with more probabilities to a satisfying minimum.

# 4 Results

## 4.1 GP Offline

### 4.1.1 Known Trajectory

As we expect, increasing the number of active points, the model performs better on a trajectory that belong to the data set used for the training step. The more active points we add, the more both the error and the variance will decrease.

active points = 10;

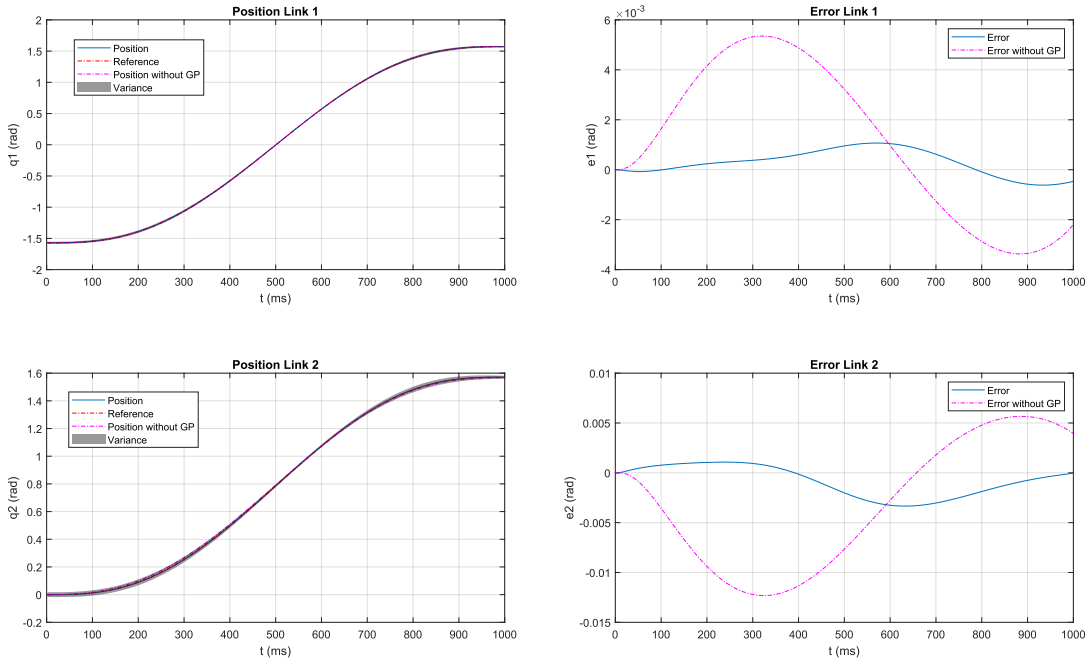


Figure 5: Plot of Position and Error.

active points = 50;

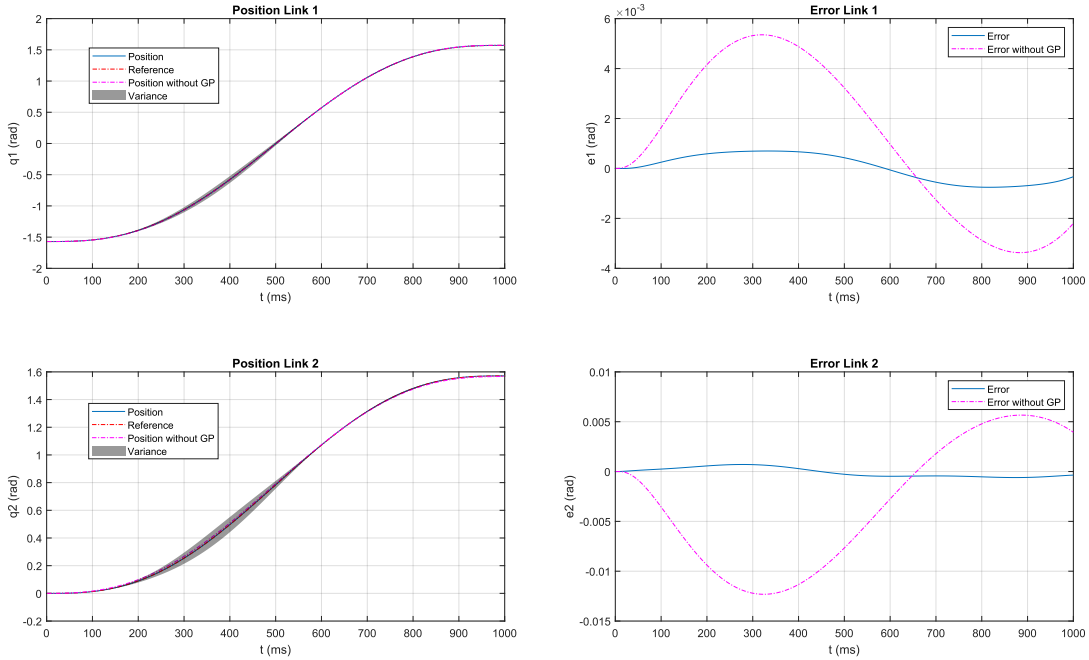


Figure 6: Plot of Position and Error.

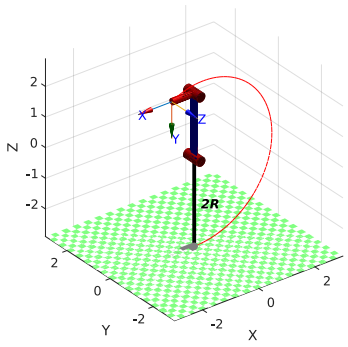


Figure 7: Plot 3D

### 4.1.2 Unknown Trajectory

Conversely, if we try to execute a trajectory which does not belong to the data set we face the exact opposite situation. Increasing the number of active points we fit the model on the training set, and so when it encounters an unseen trajectory, even if the error might decrease the variance will surely soar.

active points = 10;

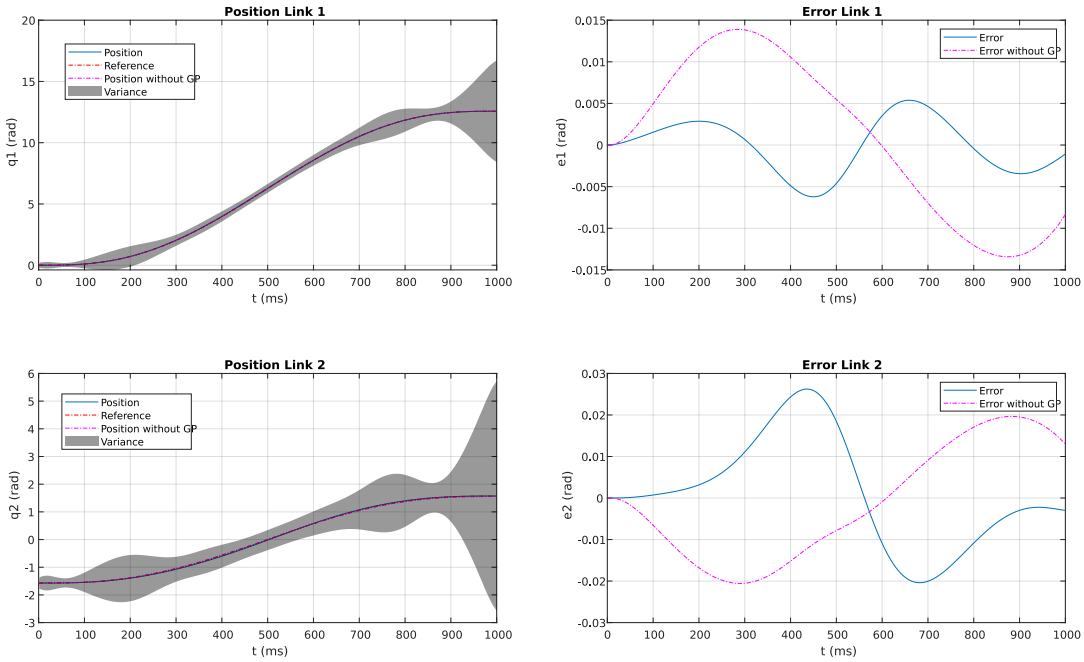


Figure 8: Plot of Position and Error.

active points = 50;

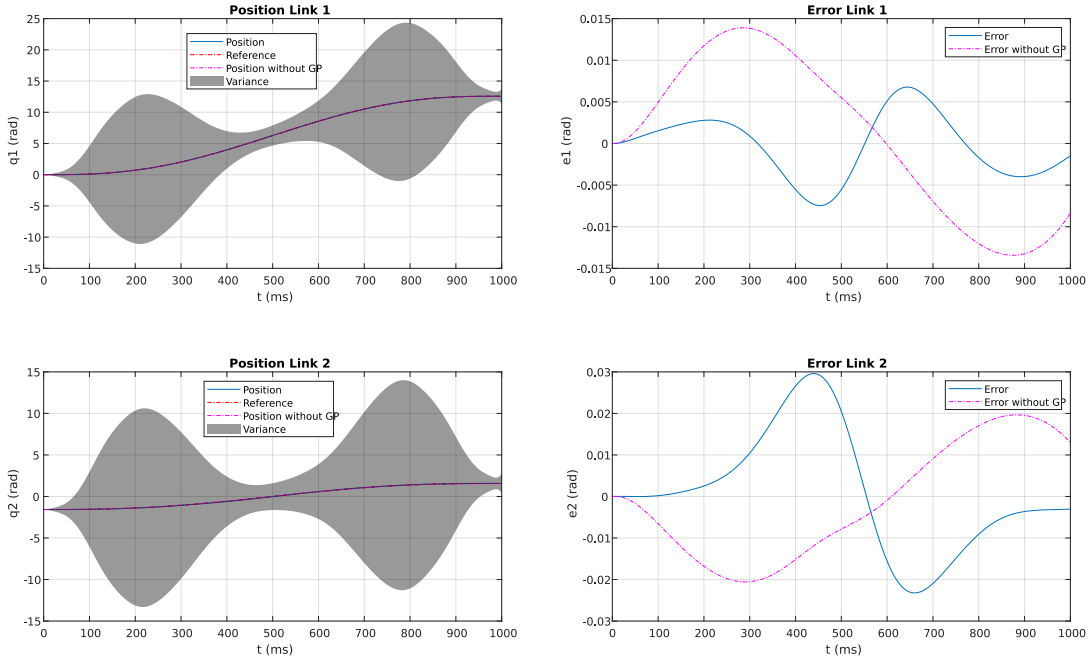


Figure 9: Plot of Position and Error.

### 4.1.3 Active Points Comparison

For the sake of completeness we compare the results between the Gaussian Process trained using a random number of active points, in place of using the heuristics to select them, maintaining the same number of active points. Obviously the results will be significantly worse in this case.

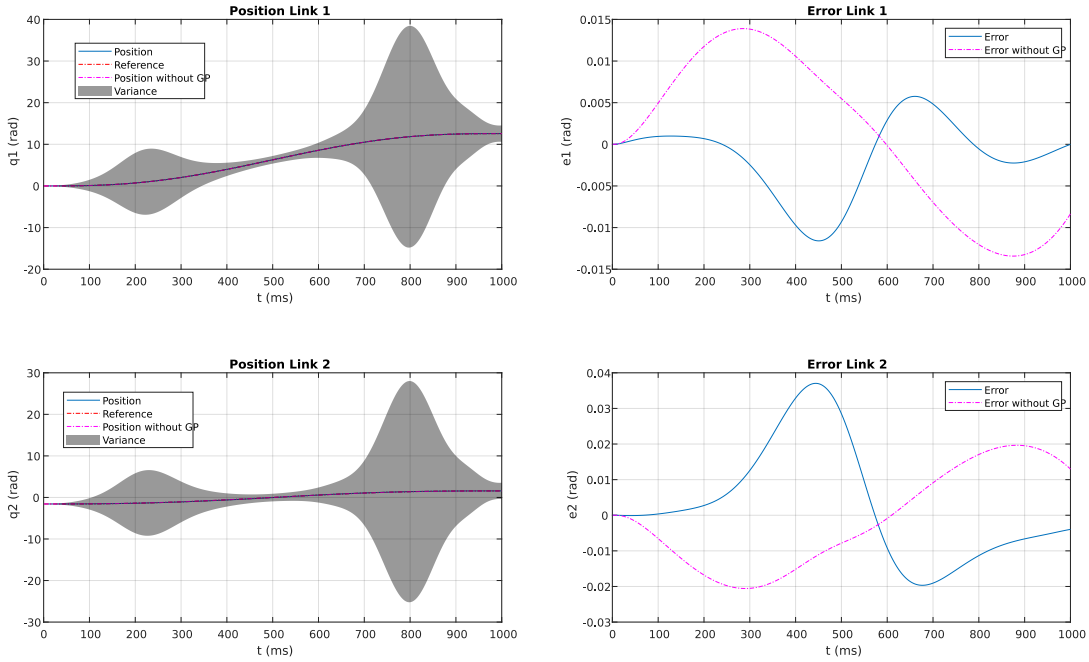


Figure 10: Plot of Position and Error.

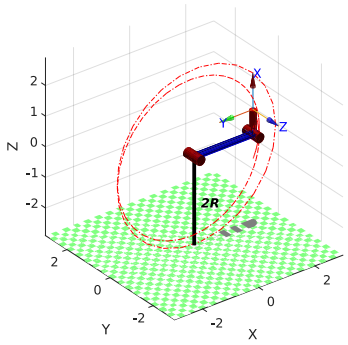


Figure 11: Plot 3D

## 4.2 GP Online

For what concerns the online training we have a different situation. As a matter of fact the number of active points does not affect in a noticeable way the performance of the Process. The reason can be found in the own structure of the training step, which, differently from the offline training, that computes the optimization step once time on the entire data set, optimizes continuously on the previous bunch of points, and the result is that the error and the variance remain much the same.

active points = 10;

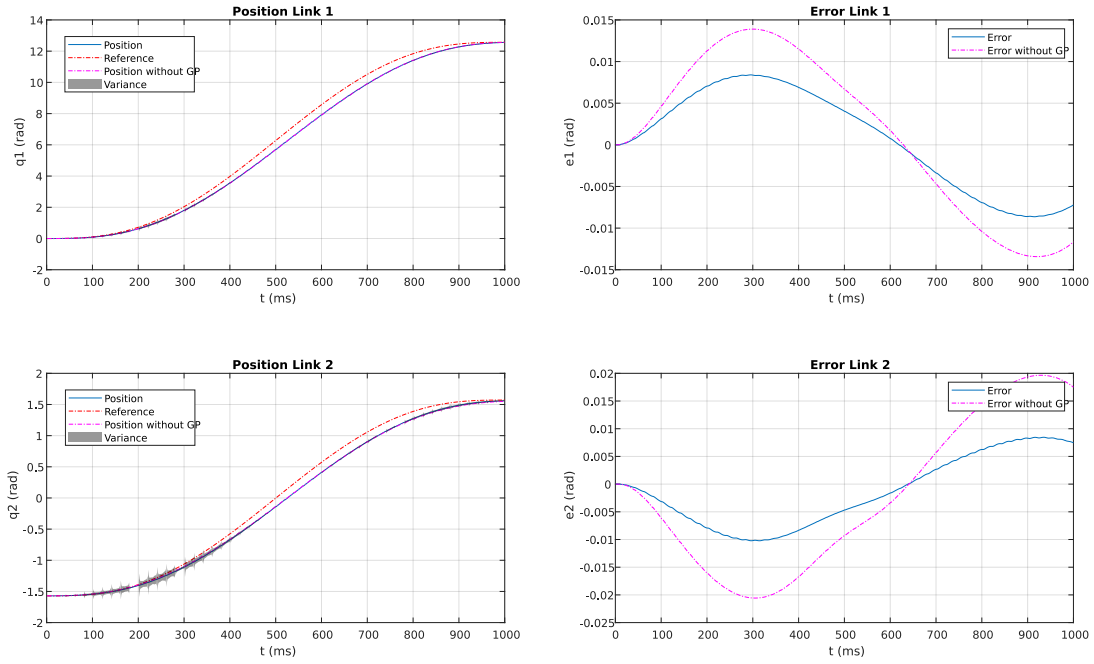


Figure 12: Plot of Position and Error.

active points = 50;

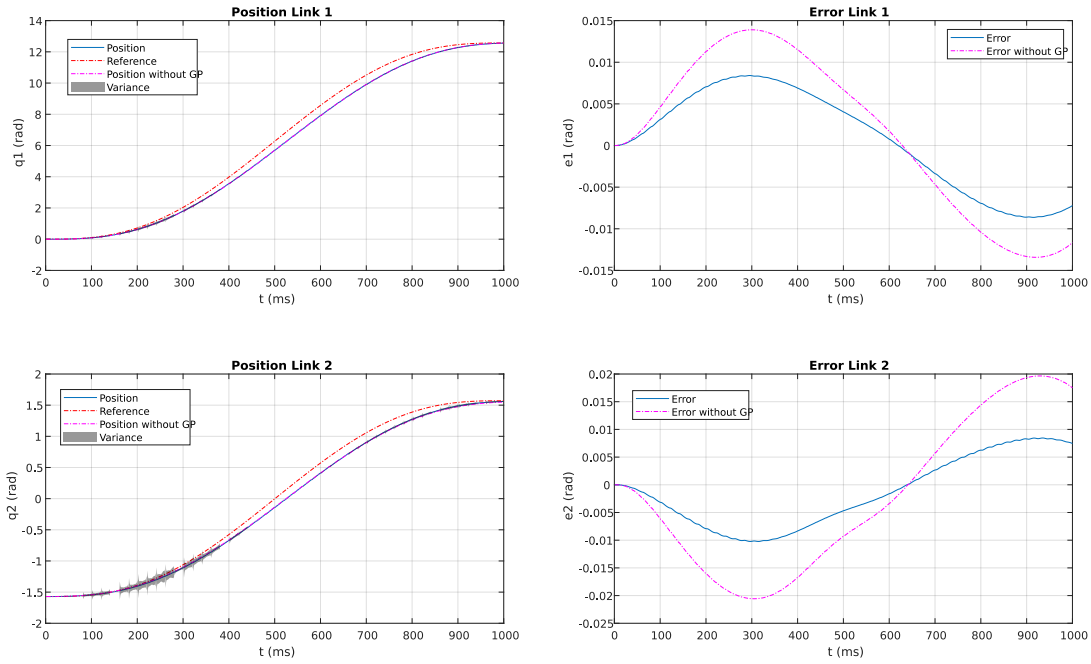


Figure 13: Plot of Position and Error.

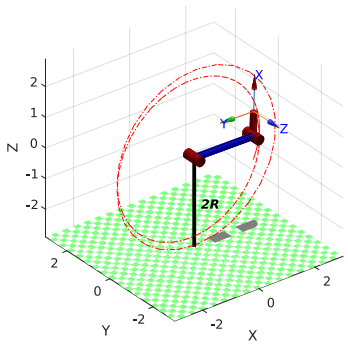


Figure 14: Plot 3D

# 5 Conclusions

As we have shown in the simulation results, Gaussian Process Regression it is a very powerful tool that allows to compensate every kind of model mismatch. Through its implementation we were been able to bring the errors between the nominal model and the real one close to zero, or even cancel them out. This means that after the computation of the described process it is possible to override the existing differences between the two models and it is possible to control the robot in a very accurate way, avoiding problems such as the need for a high gain PID controllers or simply executing high precision tasks that were really difficult to compute otherwise.
Eventually, the training process (model fitting) will lead to a model which will be perfectly tuned for predicting the torque inputs needed to execute trajectories "close" to the ones that have been used for the training step. However, the more the new desired trajectory will be far from points in the data set, the more the error will be high, that is due to the intrinsic weakness of Gaussian Process regarding outliers. Consequently if the desired trajectory has to pass by inside an already explored region it will be executed very accurately, conversely, if it contains a lot of "unseen points" the error will be higher. This "generalization problem" can be intuitively addressed by trying to explore the entire configuration space and its time derivatives, since this sparse version of the Gaussian Process is effectively capable of dealing with very large data sets.
It would be interesting to extend this model to more complex robot manipulators and perhaps to improve it by deploying more powerful covariance functions.

# References


[1] D. Nguyen-Tuong, J. Peters, and M. Seeger, "Computed torque control with nonparametric regression models", Proc. American Control Conf., pp. 212-217, 2008.

[2] D. Nguyen-Tuong and J. Peters, "Incremental sparsification for real-time online model learning", Proc. 13th Int. Conf. on Artificial Intelligence and Statistics, vol. 9, pp. 557564, 2010.

[3] J. Schreiter, D. Nguyen-Tuong, M. Toussaint, "Efficient sparsification for Gaussian process regression", 2016.

[4] D. Nguyen-Tuong, J. Peters, and M. Seeger, "Model Learning with Local Gaussian Process Regression", 2009.

[5] "Robot Learning", Chapter 15 in Springer Handbook of Robotics, 2016.

[6] C. E. Rasmussen & C. K. I. Williams, "Gaussian Processes for Machine Learning", the MIT Press, 2006.

[7] https://it.mathworks.com/products/robotics.html

[8] "Robotics 2 course material", A. De Luca