POLITECNICO DI TORINO

Master's degree course
in MECHATRONIC ENGINEERING

Master's Degree Thesis

Vibration compensation for robotic manipulators by iterative learning control

Supervisor
Prof. Michele Taragna

Candidates
Davide Cannizzaro
Stefano Votano

Company Tutor
Comau S.p.A.
Dott. Eliana Giovannitti

A.A. 2017/2018

Abstract

The elasticity of the mechanical joints used in industrial manipulators induces unwanted vibrations in the structure with consequent errors in positioning during machining operations.
The proposed thesis work is aimed at analysing and improving the performance of the manipulator in repetitive applications, in which positioning accuracy is particularly important, using as strategy the iterative learning control (ILC).
Different algorithms are analysed in terms of stability and robustness, also comparing the difficulty of implementation and the time consuming. In particular, the type of ILC developed are: PD-Type ILC, inverse plant ILC and Data Driven ILC.
The different implementations are done both for SISO and MIMO system. In the first case the plant is represented by the 4^{th} link of the Comau robot *NJ4 220 - 2.4*, while in the second one a 6-DOF industrial manipulator is considered, that is the Comau robot *Racer 7 - 1.4*. Both the systems are simulated through mathematical models in Simulink environment. The development of ILC is done both on MATLAB and Simulink environments.
The results obtained show that all the techniques adopted have positive and negative aspects, but in general all of them revealed promising improvements of the performances.

Contents

Introduction

Origin of ILC

The concept of iterative learning control (ILC) suddenly began to flourish in 1984, motivated by robots doing the same tasks many times. Arimoto et al. [1], Casalino and Bartolini [2] and Craig [3] are independent developments of similar ideas in 1984, with Uchiyama [4] being one of the precursors in 1978. Middleton et al. [5] in 1985, submitted in 1984, was another independent development motivated by robotics, but using repetitive control.
The commonly documented starting point for ILC is Arimoto et al. [1], which considered a simple first order linear servomechanism system for a voltage-controlled dc-servomotor. In the opening paragraphs a fundamental analogy between ILC and human learning is written:

> "It is human to make mistakes, but it is also human to learn from such experience. Is it possible to think of a way to implement such a learning ability in the automatic operation of dynamic systems?"

How it works

> "A basketball player shooting a free throw from a fixed position can improve his ability to score by practicing the shot repeatedly. During each shot, the basketball player observes the trajectory of the ball and consciously plans an alteration in the shooting motion for the next attempt. As the player continues to practice, the correct motion is learned and becomes ingrained into the muscle memory so that the shooting accuracy is iteratively improved. The converged muscle motion profile is an open-loop control generated through repetition and learning. This type of learned open-loop control strategy is the essence of ILC" [6]

The iterative learning control problem considers a control task that has to perform a repetitive and specific tracking command. Between each command application, the system is returned to the same initial position.

Each completion or execution of the task is described in general in the ILC literature as a pass, trial or iteration. Once an iteration is finished, all data used and generated during its accomplishment are available to compute the control action to be applied in the next iteration (see Figure 1).

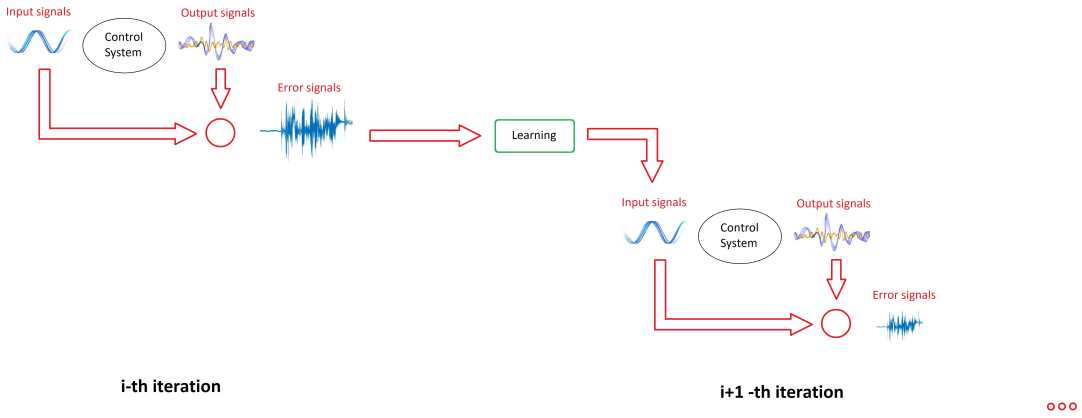

Figure 1: ILC application to a control system. Updating the input signals applied to the control system, decreasing error signals are obtained iteration by iteration.

The use of previous iteration data is a form of learning and is the base of ILC, embedding the mechanism through which performance may be improved by experience.

Difference with other learning algorithms

In literature there are many learning-type control strategies to improve the performance of classical feedback control systems, some of which similar to ILC and other very different. The main examples could be the following ones:

- Adaptive control: this kind of strategy modifies a system, the controller, while ILC generally modifies a control input or a reference, which are signals. Additionally, adaptive controllers typically do not take advantage of the information contained in repetitive signals [7].
- Machine learning control: similarly to adaptive control techniques, neural network learning involves the modification of controller parameters rather than a signal; in this case, large networks of nonlinear neurons are modified. These large networks require extensive training data but their applications are very common in non-linear control. Moreover, fast convergence may be difficult to guarantee, whereas ILC usually converges adequately in just a few iterations [8].
- Repetitive Control (RC): maybe the most similar to ILC technique, except that RC is intended for continuous operation, whereas ILC is intended for discrete operation [9].

For instance, an ILC application might be to control a robot that performs a task, returns to its initial position and comes to a rest for a certain interval of time before repeating the task. On the other hand, an RC application might be to control a hard disk drives read/write head, in which each iteration is a full rotation of the disk, and the next iteration immediately follows the current one. The difference between RC and ILC is the setting of the initial conditions for each trial. In ILC, the initial conditions are set to the same value on each iteration. In RC, the initial conditions are set to the final conditions of the previous trial.

Why ILC

The case of study, that can be extended in general to a very large number of similar situations, involves an industrial manipulator that performs a repetitive application in which, mainly due to the elasticity of the mechanical joints, unwanted vibrations in the structure induces errors in positioning during machining operations.
In this context, feedback control tracking error (see Figure 2) comes from several sources:

- deterministic and repeatable errors in following general tracking commands;
- deterministic disturbances that occur each time the same command is given (e.g. gravity on a robot link that follows a specific trajectory through the workspace);
- random disturbance errors.

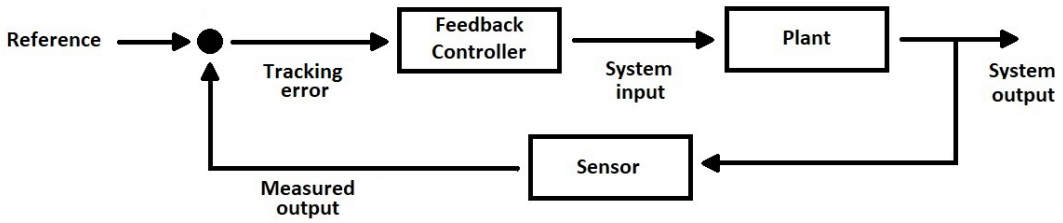

Figure 2: Feedback control system.

The aim of the ILC methods is to eliminate as much as possible of the deterministic errors. As highlighted in the previous paragraph, the choice of this learning strategies is to be preferred respect to adaptive control and machine learning control because of the repetitiveness and simplicity.
The ILC mode of operation already outlined is the most common: complete a trial, reset and then repeat. So, for what concern the scope of this thesis, it is more suitable than RC. ILC has also several advantages over a well-designed feedback and feedforward controller

(Figure 3). In particular, as discussed in [6] [7], the main benefits are in terms of rejection of repetitive disturbances, robustness respect to system model uncertainties and anticipatory behaviour due to intrinsically noncausality. The motivations are briefly resumed here for completeness.

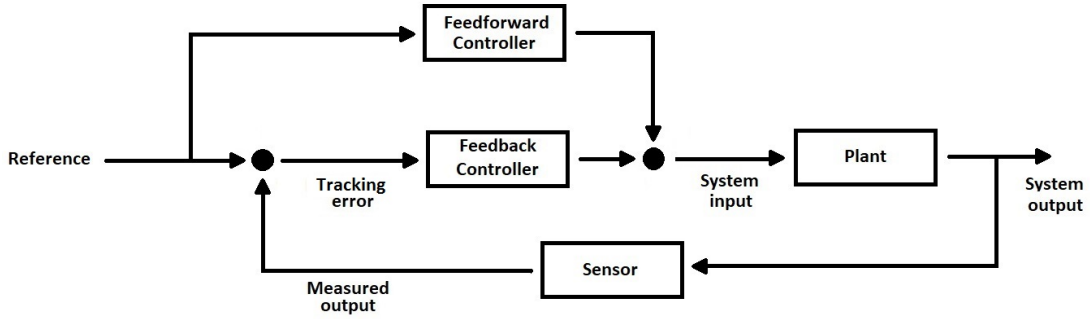

Figure 3: Feedback/feedforward control system.

> "Feedback controller reacts to inputs and disturbances and, therefore, always has a lag in transient tracking. Feedforward control can eliminate this lag, but only for known or measurable signals, such as the reference, and typically not for disturbances. ILC is anticipatory and can compensate for exogenous signals, such as repeating disturbances, in advance by learning from previous iterations. ILC does not require that the exogenous signals (references or disturbances) be known or measured, only that these signals repeat from iteration to iteration.[…] A feedback controller can accommodate variations or uncertainties in the system model. A feedforward controller instead, performs well only to the extent that the system is accurately known: friction, unmodeled nonlinear behaviour and disturbances can limit its effectiveness. Because ILC generates its open-loop control through practice (feedback in the iteration domain), this high-performance control is also highly robust to system uncertainties. Indeed, ILC is frequently designed assuming linear models and applied to systems with nonlinearities yielding low tracking errors, often on the order of the system resolution."
> [6, Iterative Learning Control Versus Good Feedback and Feedforward Design]

> "ILC has over traditional feedback and feedforward control the possibility to anticipate and pre-emptively respond to repeated disturbances. This ability depends on the causality of the learning algorithm"
> [6, Causal and Noncausal Learning]

Unlike the usual notion of noncausality, that make many techniques non-physically realizable, a noncausal learning algorithm is implementable in practice because the entire time

sequence of data is available from all previous iterations.
However, ILC cannot provide perfect tracking in every situation. Most notably, noise and nonrepeating disturbances make worse ILC performance. As with feedback control, observers can be used to limit noise sensitivity, although only to the extent to which the plant is known.
Nevertheless, unlike feedback control, the iteration-to-iteration learning of ILC provides opportunities for advanced filtering and signal processing.
To reject nonrepeating disturbances and improve the performance despite the noise, a feedback controller used in combination with the ILC is the best approach.

A lot of ILC

A first classification of different types of ILC techniques can be done considering the signal which is applied when it is combined with a feedback loop: in the serial architecture [10] the ILC control input is applied to the reference before the feedback loop, while in the parallel architecture [11] the ILC control input and feedback control input are combined. The first type of architecture (Figure 4) is useful when ILC is applied to a pre-existing system that does not allow a direct access to the controller and the control input.

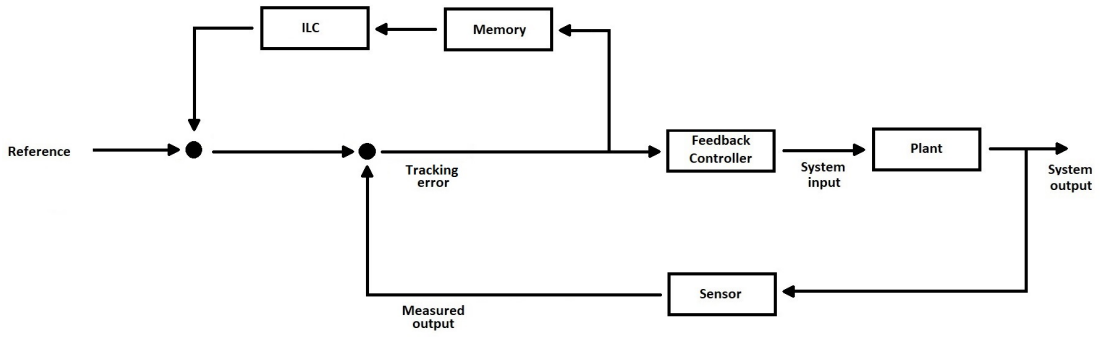

Figure 4: Example of serial ILC architecture integrated in a feedback control system.

The second type of architecture (Figure 5) is like a feedforward signals directly added to the system input in order to improve the performance and reduce the effort applied by the feedback controller when ILC converges.

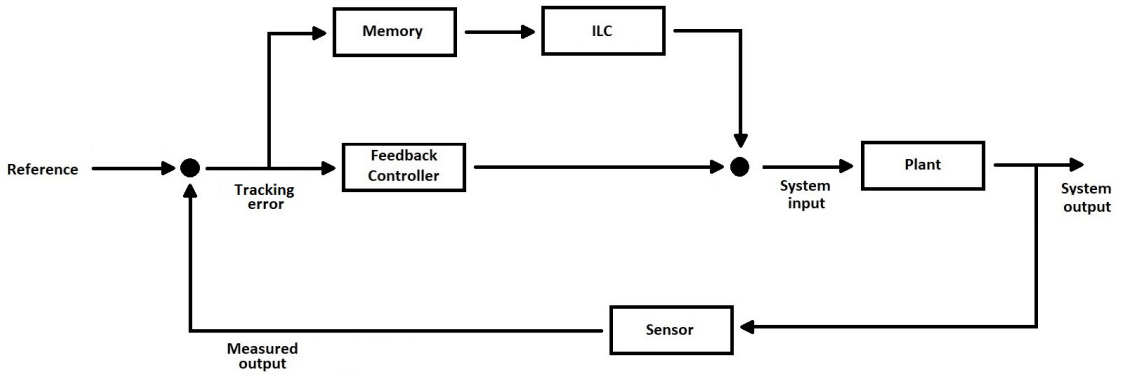

Figure 5: Example of parallel ILC architecture integrated in a feedback control system.

Another classification can be done by the different learning algorithms that are used to develop the ILC, that in general can be split up in the following classes:

- PD-ILC: the ILC control input is proportional to the error and/or the first derivate of the error, similarly to an open-loop PD controller. The advantage is clearly the simplicity, but not in all the systems is possible to find a convergent solution [5][12].

- Inverse plant ILC: the inverse of the transfer function of the plant is used as learning filter. Neglecting that the model of the plant is just an approximation of the real plant, the main difficulty of this technique is due to the fact that leads to the inversion of a non-minimum phase system [13][14].

- Robust ILC: through classical H_∞ synthesis [15] or other techniques based on current iteration [16], a learning filter that offers the fastest convergence given the model of the control system is designed. Then, usually μ-analysis approach is used to test its robustness.

- Optimal ILC: the learning filter is designed in order to minimize a performance criterion, e.g. in Q-ILC (Quadratically Optimal ILC) a quadratic next iteration cost functional is minimized [17].

- Data Driven-ILC (DD-ILC): the control input is directly computed through measurable signals avoiding the design of a learning filter [18]. Differently from other algorithms, higher computation power and data reliability is required. On the contrary of other classes of ILC, model system uncertainties do not hinge its performance.

The distinctions between different classes is not always possible. In literature, especially in these last years, a lot of "hybrids" ILC have been developed, taking the positive aspect of different classes and bringing together in a unique ILC algorithm, e.g. dual-stage robust ILC [19].

Thesis Outline

Chapter 1 and 2 describes essentially ILC approaches developed during the thesis project work: PD-ILC, Plant inverse ILC and Data Driven ILC. Some theorems about ILC stability are also shown.
In Chapter 3 and 4 are shown respectively the first (a simplified SISO system of the Comau robot *NJ4 220 - 2.4 – 4^{th}* link) and second (complete MIMO system of the Comau robot *Racer 7 – 1.4*) simulator given by Comau, presenting ILC modification required and the obtained results.
Conclusions and some recommendations for further work are given in Chapter 5.

Chapter 1

System Characteristic

In the following sections, the system to which we refer during the thesis and its representations are described, then some criterions for stability, robustness and performance are analysed.

1.1 System description

An LTI discrete-time SISO system is considered, but it can be easily generalized for a MIMO system:

$$y_j(k) = P(q)u_j(k) + d(k) \tag{1.1}$$

where j is the iteration index, k is the time index and q is a forward time-shift operator $qx(k) \equiv x(k+1)$, u_j is the control input, y_j is the output, and d a repeating disturbance. $P(q)$ is a proper function in q that describes the overall system with its delay of order m; to apply the ILC $P(q)$ must be asymptotically stable so the closed-loop system is considered. A N-sample sequence of inputs and outputs is collected

$$\begin{aligned} &u_j(k), k \in \{0, 1, \dots, N-1\}, \\ &y_j(k), k \in \{m, m+1, \dots, N+m-1\}, \\ &d(k), k \in \{m, m+1, \dots, N+m-1\}, \end{aligned}$$

and the desired system output

$$y_d(k), k \in \{m, m+1, \dots, N+m-1\}.$$

The error signal is calculated as $e_j(k) = y_d(k) - y_j(k)$. In some case is better for analysis and design to consider N and the iteration index as infinite [20]. The plant delay, that depends by the system considered, is assumed to be $m = 1$ for simplicity.
Because ILC requires the storage of past iterations, the domain typically used is the discrete time. System (1.1) is quite general to consider IIR (Infinite Impulse Response) and FIR (Finite Impulse Response) system. Repeating disturbances, repeated nonzero

initial conditions and similar disturbances can be captured by $d(k)$.
A very common ILC learning algorithm [7] is

$$u_{j+1}(k) = Q(q)[u_j(k) + L(q)e_j(k+1)] \tag{1.2}$$

where $L(q)$ and $Q(q)$ are the learning function and the Q-filter.
The scheme of the ILC function (1.2) and the plant dynamics (1.1) are shown in Figure 1.1 [6].

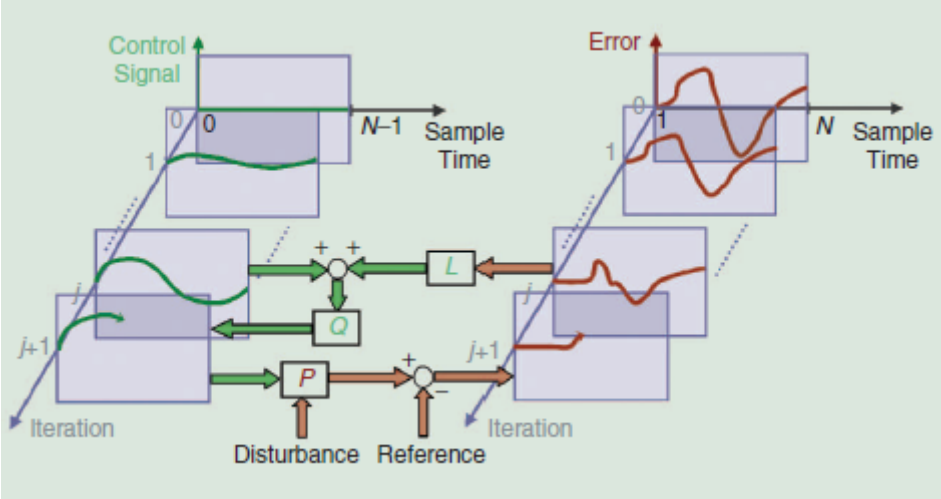

Figure 1.1: ILC scheme. The error is filtered first through L and then Q to obtain the input to the plant P.

1.2 System representation

One possible representation of the system is the lifted-system: to build it the LTI plant (1.1) is expanded as an infinite power series obtaining

$$P(q) = p_1 q^{-1} + p_2 q^{-2} + p_3 q^{-3} + \cdots \tag{1.3}$$

where the coefficients p_k are the Markov parameters [21].
From the state space description

$$\begin{aligned} \mathbf{x}_j(k+1) &= \mathbf{A}\mathbf{x}_j(k) + \mathbf{B}u_j(k) \\ y_j(k) &= \mathbf{C}\mathbf{x}_j(k) \end{aligned}$$

p_k is given by $pk = \mathbf{C}\mathbf{A}^{k-1}\mathbf{B}$, because $m = 1$ then $p1 \neq 0$. From this representation the system can be described using an $N \times N$-dimensional lifted form

$$\underbrace{\begin{bmatrix} y_j(1) \\ y_j(2) \\ \vdots \\ y_j(N) \end{bmatrix}}_{\mathbf{y}_j} = \underbrace{\begin{bmatrix} p_1 & 0 & \cdots & 0 \\ p_2 & p_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ p_N & p_{N-1} & \cdots & p_1 \end{bmatrix}}_{\mathbf{P}} \underbrace{\begin{bmatrix} u_j(0) \\ u_j(1) \\ \vdots \\ u_j(N-1) \end{bmatrix}}_{\mathbf{u}_j} + \underbrace{\begin{bmatrix} d(1) \\ d(2) \\ \vdots \\ d(N) \end{bmatrix}}_{\mathbf{d}} \tag{1.4}$$

and

$$\underbrace{\begin{bmatrix} e_j(1) \\ e_j(2) \\ \vdots \\ e_j(N) \end{bmatrix}}_{\mathbf{e}_j} = \underbrace{\begin{bmatrix} y_d(1) \\ y_d(2) \\ \vdots \\ y_d(N) \end{bmatrix}}_{\mathbf{y}_d} - \underbrace{\begin{bmatrix} y_j(1) \\ y_j(2) \\ \vdots \\ y_j(N) \end{bmatrix}}_{\mathbf{y}_j} \tag{1.5}$$

A delay $m = 1$ is considered to ensure that all the diagonal entries are not zero. For a generic system with a delay of m, the lifted representation is

$$\begin{bmatrix} y_j(m) \\ y_j(m+1) \\ \vdots \\ y_j(m+N-1) \end{bmatrix} = \begin{bmatrix} p_m & 0 & \cdots & 0 \\ p_{m+1} & p_m & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ p_{m+N-1} & p_{m+N-2} & \cdots & p_m \end{bmatrix} \begin{bmatrix} u_j(0) \\ u_j(1) \\ \vdots \\ u_j(N-1) \end{bmatrix} + \begin{bmatrix} d(m) \\ d(m+1) \\ \vdots \\ d(m+N-1) \end{bmatrix}$$

$$\begin{bmatrix} e_j(m) \\ e_j(m+1) \\ \vdots \\ e_j(m+N-1) \end{bmatrix} = \begin{bmatrix} y_d(m) \\ y_d(m+1) \\ \vdots \\ y_d(m+N-1) \end{bmatrix} - \begin{bmatrix} y_j(m) \\ y_j(m+1) \\ \vdots \\ y_j(m+N-1) \end{bmatrix}$$

The lifted form (1.4) allows us to easily write the SISO system as a MIMO system if necessary. The time-domain dynamics are contained in the $\mathbf{P}$ structure, y_j, u_j and d are contained in vectors $\mathbf{y}_j$, $\mathbf{u}_j$ and $\mathbf{d}$.
In a similar way the Q-filter and the learning function can be written in lifted form considering that they can be non-casual functions

$$Q(q) = \cdots + q_{-2}q^2 + q_{-1}q^1 + q_0 + q_1q^{-1} + q_2q^{-2} + \cdots$$

and

$$L(q) = \cdots + l_{-2}q^2 + l_{-1}q^1 + l_0 + l_1q^{-1} + l_2q^{-2} + \cdots$$

So (1.2) becomes

$$\underbrace{\begin{bmatrix} u_{j+1}(0) \\ u_{j+1}(1) \\ \vdots \\ u_{j+1}(N-1) \end{bmatrix}}_{\mathbf{u}_{j+1}} = \underbrace{\begin{bmatrix} q_0 & q_{-1} & \cdots & q_{-(N-1)} \\ q_{-1} & q_{-2} & \cdots & q_{-(N-2)} \\ \vdots & \vdots & \ddots & \vdots \\ q_{N-1} & q_{N-2} & \cdots & q_0 \end{bmatrix}}_{\mathbf{Q}} \times \left(\underbrace{\begin{bmatrix} u_j(0) \\ u_j(1) \\ \vdots \\ u_j(N-1) \end{bmatrix}}_{\mathbf{u}_j} + \underbrace{\begin{bmatrix} l_0 & l_{-1} & \cdots & l_{-(N-1)} \\ l_{-1} & l_{-2} & \cdots & l_{-(N-2)} \\ \vdots & \vdots & \ddots & \vdots \\ l_{N-1} & l_{N-2} & \cdots & l_0 \end{bmatrix}}_{\mathbf{L}} \underbrace{\begin{bmatrix} e_j(1) \\ e_j(2) \\ \vdots \\ e_j(N) \end{bmatrix}}_{\mathbf{e}_j} \right) \tag{1.6}$$

When $L(q)$ and $Q(q)$ are causal functions their matrix representation becomes lower triangular, because the system is LTI all the entries along each diagonal are identical. So the matrix are Toeplitz [22].
Another representation is the frequency-domain, that is obtained through the z-transform by replacing q with z. To apply the z-transformation to the ILC (1.1), (1.2), the number of samples N must be $N \to \infty$; in practical applications of ILC the trial duration is finite, so the z-domain representation is approximated. The transformed representation of the previous equations (1.1), (1.2) are

$$Y_j(z) = P(z)U_j(z) + D(z) \tag{1.7}$$

and

$$U_{j+1}(z) = Q(z)[U_j(z) + zL(z)E_j(z)] \tag{1.8}$$

where $E_j(z) = Y_d(z) - Y_j(z)$. For a generic m time-step delay, z^m is used instead of z in (1.8).

1.3 Analysis and theorems

For an ILC system is important to guarantee the stability. System (1.1), (1.2) are asymptotically stable (AS) if exist $\bar{u} \in \mathbb{R}$ such that

$$|u_j(k)| \leq \bar{u}, \forall k \in \{0, 1, \dots, N-1\} \text{ and } j \in \{0, 1, \dots, \}$$

and $\forall k \in \{0, 1, \dots, N-1\}$

$$\exists \lim_{j \to \infty} u_j(k)$$

The converged control input is defined as $u_\infty(k) = \lim_{j\to\infty} u_j(k)$. The following conditions for AS are developed in [20]. Substituting (1.5) and (1.4) in (1.6) is obtained

$$\mathbf{u}_{j+1} = \mathbf{Q}(\mathbf{I} - \mathbf{LP})\mathbf{u}_j + \mathbf{QL}(\mathbf{y}_d - \mathbf{d}) \tag{1.9}$$

If $\rho(\mathbf{A}) = max_i\,|\lambda_i(\mathbf{A})|$ is considered as the spectral radius of the matrix $\mathbf{A}$, and $\lambda_i(\mathbf{A})$ the i^{th} eigenvalue of $\mathbf{A}$, the following condition is obtained that is necessary and sufficient for stability [20]

$$\rho(\mathbf{Q}(\mathbf{I}-\mathbf{LP})) < 1 \tag{1.10}$$

If the Q-filter and L are casual, the matrix $\mathbf{Q}(\mathbf{I}-\mathbf{LP})$ is lower triangular with multiple eigenvalues

$$\lambda = q_0(1-l_0p_1) \tag{1.11}$$

so 1.10 is equivalent to

$$|q_0(1-l_0p_1)| < 1 \tag{1.12}$$

Another condition to guarantee AS that is only sufficient is

$$\left\|Q(z)[1-zL(z)P(z)]\right\|_\infty < 1 \tag{1.13}$$

with an infinite number of samples. If Q and L are casual functions the system is AS also for finite-duration ILC [20].
The performance characteristic of an ILC system is based on the asymptotic value of the error.

$$\begin{aligned} e_\infty(k) &= \lim_{j\to\infty} e_j(k) \\ &= \lim_{j\to\infty} (y_d(k) - P(q)u_j(k) - d(k)) \\ &= y_d(k) - P(q)u_\infty(k) - d(k) \end{aligned}$$

To judge the performance of the algorithm the error from the first iteration can be compared to the last one or, more quantitatively, various metrics can be used like RMSE or norm-∞, norm-1, norm-2 based on which minimization is more interesting.
If the ILC system is AS, then with $N \to \infty$

$$e_\infty = [\mathbf{I}-\mathbf{P}[\mathbf{I}-\mathbf{Q}(\mathbf{I}-\mathbf{LP})]^{-1}\mathbf{QL}](\mathbf{y}_d - \mathbf{d}) \tag{1.14}$$

or similarly

$$E_\infty(z) = \frac{1-Q(z)}{1-Q(z)[1-zL(z)P(z)]}[Y_d(z) - D(z)] \tag{1.15}$$

Many ILC systems are built to converge to zero error $e_\infty(k) = 0$ for all k. If P and L are not identically zero a necessary and sufficient condition for which $e_\infty(k) = 0$ for all k is that the system (1.1), (1.2) is AS and $Q(q) = 1$. A prof of this can be found in [23] and [11].
The presence of a lowpass Q-filter can improve the robustness and transient learning.
The transient learning behaviour happens when the tracking error grows rapidly over the first iterations and decrease only after a huge number of them (e.g. > 100). In practice it is difficult to distinguish from instability because the initial growth rate and magnitude are

so large.
Some considerations can be done to avoid large learning transient [20], but, in some cases, this phenomenon can be more important then stability. Some unstable ILC can be effective when their initial transient quickly decreases the error [24], if the learning is stopped at a low error and before the system diverges, this condition can be defined as a "practical stability".
In the ILC system another key point is robustness. If the system to control is known exactly a robust ILC controller can be easily found; when the plant is uncertain, it is necessary to take into account the possible errors in modelling and also the perturbations due to the nonrepeating disturbances. To guarantee stability and good transient some conditions are required, which leads to limitations on the achievable performance of the ILC system.
A key question is whether an AS ILC system is also AS to plant perturbations. For example taking a system where $Q(q) = 1$ (so zero converged error) and $L(q)$ is casual, the stability condition (1.12) is $|1 - l_0 p_1| < 1$. Assuming l_0 and p_1 both nonzero the ILC system is AS if and only if

$$sgn(p_1) = sgn(l_0) \tag{1.16}$$

and

$$l_0 p_1 \leq 2 \tag{1.17}$$

In this example to guarantee stability only the sign of p_1 is to be considered so perturbations on other components of the plant do not destabilize the ILC system, choosing a small l_0 it can be concluded that the system is robust if the sign of p_1 does not change.
Considering the uncertain plant

$$P(q) = \hat{P}(q)[1 + W(q)\Delta(q)] \tag{1.18}$$

where $W(q)$ is known and stable, $\hat{P}(q)$ is the nominal plant model and $\Delta(q)$ is unknown and stable with $\|\Delta(z)\|_\infty < 1$.
The following condition guarantees robust monotonicity, if

$$\left|W(e^{i\theta})\right| \leq \frac{\gamma^* - |Q(e^{i\theta})|\left|1 - e^{i\theta}L(e^{i\theta})\hat{P}(e^{i\theta})\right|}{|Q(e^{i\theta})|\left|e^{i\theta}L(e^{i\theta})\hat{P}(e^{i\theta})\right|} \tag{1.19}$$

for all $\theta \in [-\pi, \pi]$, then the ILC system (1.1), (1.2), (1.18) with $N \to \infty$ is monotonically convergent with convergence rate $\gamma^* < 1$.
From equation (1.19) it can be seen that monotonic robustness depends on the dynamics of $P(q)$, $Q(q)$ and $L(q)$, the most effective and simplest way to increase robustness is to reduce the Q-filter gain at a given frequency but this impacts the converged performance, a trade-off between performance and robustness is required during the choose of the Q-filter.
Other problems that lead to instability are time-varying delay in the plant, nonrepeating disturbance, noise and initial condition variation, some solutions are discussed in [25], [26] and [27].

Chapter 2

ILC approaches

From a practical point of view, the basic idea of the ILC algorithm is to improve the response of the system to obtain an output error smaller than the result of a basic control strategy. ILC generates an open-loop signal that is used to refine the tracking of the reference and reject the repeating disturbance of the plant.
ILC is able to learn the repeating errors but ignores the nonrepeating disturbances because there is no feedback mechanism. As already said in the introduction, a possible solution is to use the ILC control in combination with other control algorithms that can work out with nonrepeating disturbances, for example using feedback-feedforward controller.
In the following sections, we talk over three ILC algorithms developed during the thesis: the PD-type, the Plant Inversion and the Data Driven. Some considerations are done about robustness, performance and convergence.

2.1 PD-Type and Tunable Designs

The PD-type is a learning function with tunable parameters. It is a combination of simplest learning functions like D-type or P-type used in Arimoto's work [1], these types of algorithms are widely used in particular for nonlinear systems [1], [28], [29], [30].
This type of ILC can be applied to a system without a heavy modelling and analysis, so it is the simplest algorithm. It is similar to a PID feedback control but in ILC the integrator term I is rarely used because ILC has already an integrator behaviour from one trial to the next.
The PD-type learning function in discrete-time can be written as:

$$u_{j+1}(k) = u_j(k) + k_p e_j(k+1) + k_d[e_j(k+1) - e_j(k)] \tag{2.1}$$

From (1.10), the ILC system is AS if and only if $|1 - (k_p + k_d)p_1| < 1$. When p_1 is knows is possible to find k_p and k_d such that the ILC system is AS. With PD-type learning is difficult to achieve monotonic convergence but if the iterations are sufficiently short a solution could be found [31].

A possible solution to the problem of monotonic convergence is achieved modifying the learning algorithm to include a lowpass Q-filter [9], [32], [33], in order to disable the learning at high frequency.
In PD-type ILC the most common method for selecting the gains k_p and k_d is by tuning [1], [34], [35], choosing the type and order of the Q-filter, the cutoff frequency becomes a tuning variable. Despite the popularity of this approach, ILC tuning guidelines are not available: the goals of the tuning is to achieve a good learning transient and a low error therefore to find the best solution for each set of gains k_p and k_d the learning is reset and run for sufficient iterations. When a stable baseline of parameters is found, they can be increased but, depending on the application, the gains influence the rate of convergence, whereas the Q-filter influences the performance. The Q-filter bandwidth is selected to satisfy the stability condition: a large bandwidth increases the performance but decreases the robustness.

2.2 Plant Inversion Methods

In the plant inversion methods the learning function is a model of the inverted plant dynamics. The inversion ILC algorithm in discrete-time is

$$u_{j+1}(k) = u_j(k) + \hat{P}^{-1}(q)e_j(k). \tag{2.2}$$

Then rewriting it as $u_{j+1}(k) = u_j(k) + q^{-1}\hat{P}^{-1}(q)e_j(k+1)$ it can be seen that the learning function is $L(q) = q^{-1}\hat{P}^{-1}(q)$, which is casual and has zero relative degree.
Assuming that $P(q)$ is an exact model of the plant, the convergence occurs in just one iteration and the converged error is $e_\infty \equiv 0$. Due to plant uncertainty the previous conditions is difficult to reach.
One of the most difficult problem with plant inversion is dealing with nonminimum phase system in which a direct inversion leads to an unstable filter. The use of an unstable filter generates undesirably large control signals. To avoid this problem a stable inversion approach could be used, which leads to a noncausal learning function [13], [14]. If the system is nonlinear, find a stable inverse filter could be difficult but in some case the inversion of the system dynamics linearized in the operating point could be sufficient to achieve a good result [32].
Whatever it is the type of $P(q)$, the goodness of the plant inversion depends on the accuracy of the model used. A mismatch between the nominal plant and the real dynamics prevents convergence in one iteration and could lead to a poor transient behaviour. If at a given frequency the system has an uncertainty greater than 100%, it is not robustly monotonically convergent and so it is not the plant inversion algorithm. To avoid poor transients due to model uncertainty a lowpass Q-filter is employed [32]; the cutoff frequency is set sufficiently low to disable the learning of high frequencies with a large uncertainty and to achieve robust monotonicity.

2.3 Data Driven ILC

Iterative learning control algorithms such as Inverse Plant ILC and Optimal ILC are intrinsically model-based. The convergence and performance properties of these learning control algorithms are strongly dependent on a model of the controlled system. In particular, robustness to modelling errors is a key issue, as is evidenced by the development of robust ILC approaches such as [15] [16].
Robust ILC approaches require both a nominal model and a description of model uncertainty. Especially in the multivariable situation, such models are difficult and expensive to obtain. The aim of this chapter is to report an optimal ILC algorithm for multivariable systems without the need of a model to design L and Q filters [18].

2.3.1 Preliminaries

A single-input single-output (SISO) system J^{11} transfer function can be expressed by:

$$J^{11}(z) = \sum_{i=0}^{\infty} h_i z^{-i} \tag{2.3}$$

where $h_i \in \mathbb{R}$ $i = 0, ..., \infty(z)$ are the Markov parameters of J^{11}, and $z \in \mathbb{C}$. It is assumed that signals have finite length $N \in \mathbb{N}$.
More in general, for a multiple-input multiple-output (MIMO) system J with transfer function matrix $J(z) \in \mathbb{C}^{n_o \times n_i}$ (with n_i the number of inputs and n_o the number of outputs), the finite-time response is denoted as

$$\underbrace{\begin{bmatrix} y^1 \\ \vdots \\ y^{n_o} \end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix} J^{11} & \cdots & J^{1,n_i} \\ \vdots & \ddots & \vdots \\ J^{n_o,1} & \cdots & J^{n_o,n_i} \end{bmatrix}}_{\mathbf{J}} \underbrace{\begin{bmatrix} u^1 \\ \vdots \\ u^{n_o} \end{bmatrix}}_{\mathbf{u}} \tag{2.4}$$

where J^{ij} is the matrix representation of the ij^{th} entry in $J(z)$, $y \in \mathbb{R}^{n_o N}$, $u \in \mathbb{R}^{n_i N}$ and $J \in \mathbb{R}^{n_o N \times n_i N}$ as matrix representation of $J(z)$.

2.3.2 Optimal adjoint-based ILC

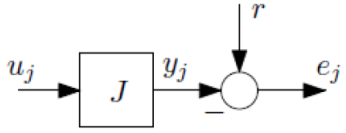

Figure 2.1: Multivariable ILC setup.

The ILC framework used is reported in Figure 2.1. The system $J \in \mathbb{R}^{n_o N \times n_i N}$is a MIMO system with output $y_j \in \mathbb{R}^{n_o N}$, input $u_j \in \mathbb{R}^{n_i N}$ and reference $r \in \mathbb{R}^{n_o N}$. The trial index is denoted as j.
From Figure 2.1 follows that the tracking errors in trial j and $j+1$ are

$$e_j = r - Ju_j \tag{2.5}$$

$$e_{j+1} = r - Ju_{j+1} \tag{2.6}$$

and by substituting (2.5) in (2.6), the error propagation from trial j to $j+1$ is obtained as

$$e_{j+1} = e_j - J(u_{j+1} - u_j) \tag{2.7}$$

In Optimal ILC class algorithms, u_{j+1} is determined by minimizing a cost function. In the proposed technique, the performance criterion $\mathcal{J}(u_{j+1})$ is given by

$$\mathcal{J}(u_{j+1}) := \left\|e_{j+1}\right\|_{W_e} + \left\|u_{j+1}\right\|_{W_f} \tag{2.8}$$

with $\|x\|_W = x^T W x$, $W_e \in \mathbb{R}^{n_o N \times n_i N}$, $W_f \in \mathbb{R}^{n_o N \times n_i N}$. W_e and W_f are respectively a positive definite and semi-definite weight matrix.
To minimize 2.7, a gradient descend (or steepest descent) algorithm is used

$$u_{j+1} = (I\epsilon W_f)u_j + \epsilon J^T W_e e_j \tag{2.9}$$

where $0 < \epsilon < \bar{\epsilon}$ is the learning gain. The proof [19, Section IIIB Theorem 2] is given computing the gradient of (2.8) at the current ILC command u_j and by performing the learning update in the steepest descent direction.
[19, Section IIIB - Lemma 4] reveals that the gradient descent ILC algorithm (2.9) can be interpreted as an adjoint-based algorithm.
For what concern the upper bound $\bar{\epsilon}$ of the learning gain ϵ, a criterion for monotonic convergence is given in [19, Section IIID Theorem 8], briefly reported below.
Let $0 < \epsilon < \bar{\epsilon}$, then $\exists \bar{\epsilon} > 0$ such that ILC algorithm (2.9) is monotonically convergent with

$$\bar{\epsilon} = 2\left\|J^T W_e J + W_f\right\|^{-1} \tag{2.10}$$

and converged signals

$$u_\infty = \lim_{j\to\infty} u_j = (J^T W_e J + W_f)^{-1} J^T W_e r$$

$$e_\infty = \lim_{j\to\infty} e_j = (I - J(J^T W_e J + W_f)^{-1} J^T W_e) r$$

The proof is based on the contraction mapping theorem, see [36, Theorem 3.15.2].

2.3.3 Data Driven learning using the adjoint sytem

An operation with the adjoint of a linear time invariant SISO system can be recast to an operation on the original system and time-reversal of the input and output signals.

$$J^{11^T} = RJ^{11}R \tag{2.11}$$

with

$$R = \begin{bmatrix} 0 & \cdots & 0 & 0 & 1 \\ 0 & \cdots & 0 & 1 & 0 \\ \vdots & & \iddots & 0 & 0 \\ 0 & \iddots & & \vdots & \vdots \\ 1 & \cdots & 0 & 0 & 0 \end{bmatrix} \tag{2.12}$$

an involutory permutation matrix with size $N \times N$. Here, R is interpreted as a time-reversal operator.

$$y = J^{11^T}u = RJ^{11}Ru \tag{2.13}$$

The time-reversal approach only applies to SISO systems.
For a MIMO system J, the adjoint J^T can be written as:

$$J^T = \underbrace{\begin{bmatrix} R & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & R \end{bmatrix}}_{\mathbf{R_{n_i}}} \underbrace{\begin{bmatrix} J^{11} & \cdots & J^{1,n_i} \\ \vdots & \ddots & \vdots \\ J^{n_o,1} & \cdots & J^{n_o,n_i} \end{bmatrix}}_{\tilde{\mathbf{J}}} \underbrace{\begin{bmatrix} R & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & R \end{bmatrix}}_{\mathbf{R_{n_o}}} \tag{2.14}$$

where $\tilde{J} \in \mathbb{R}^{n_i N \times n_o N}$, and $R_{n_i} \in \mathbb{R}^{n_i N \times n_i N}$, $R_{n_o} \in \mathbb{R}^{n_o N \times n_o N}$ are time-reversal operators for the higher dimensional input and output signals. Matrix $\tilde{J}$ is the finite-time representation of $\tilde{J}(z) \in \mathbb{C}^{n_i \times n_o}$.
The main idea presented in [18] is to develop a data-driven MIMO ILC algorithm by recasting the system $\tilde{J}(z)$ as

$$\tilde{J}(z) = \sum_{i=1}^{n_i} \sum_{j=1}^{n_o} I^{ij}(z) J I^{ij} \tag{2.15}$$

where $I^{ij} \in \mathbb{R}^{n_i \times n_o}$, is a static system with n_o outputs and n_i inputs. For the k^{th} and l^{th} entry of I^{ij} holds

$$\begin{aligned} I^{ij}_{k=i,l=j} &= 1 \\ I^{ij}_{k\neq i,l\neq j} &= 0 \end{aligned} \tag{2.16}$$

i.e., all entries of I^{ij} are zero, except the i^{th}, j^{th} entry. The structure of I^{ij} is given by

$$I^{ij} = \begin{bmatrix} 0^{i-1 \times j-1} & 0^{i-1 \times 1} & 0^{i-1 \times n_o-j} \\ 0^{1 \times j-1} & 1 & 0^{1 \times n_o-j} \\ 0^{n_i-i \times j-1} & 0^{n_i-i \times 1} & 0^{n_i-i \times n_o-j} \end{bmatrix} \tag{2.17}$$

Let I^{ij} be the finite-time representation of I^{ij}, then the finite-time representation of (2.15) is given by

$$\tilde{J} = \sum_{i=1}^{n_i} \sum_{j=1}^{n_o} I^{ij} J I^{ij} \tag{2.18}$$

and by substituting it in (2.14) can be finally obtained

$$J^T = R_{n_i} \left(\sum_{i=1}^{n_i} \sum_{j=1}^{n_o} I^{ij} J I^{ij} \right) R_{n_o} \tag{2.19}$$

So, J^T can be computed by performing $n_i \times n_o$ experiments on system J.
Data-driven ILC algorithm for MIMO systems, explained in [18, Summary 6] and used in next chapters of this work, is here reported:

"Given an initial input u_0, set $j = 0$, perform the following steps:

- execute a trial and measure $e_j = r - Ju_j$
- experimentally determine $J^T W_e e_j$ (this step can be also visualized in Figure 2.2)
  - o time reverse $\bar{e}_j = R_{n_o} e_j$
  - o compute $\bar{z}_j$ by performing $n_i \times n_o$ experiments on J

$$\bar{z}_j = \sum_{i=1}^{n_i} \sum_{j=1}^{n_o} I^{ij} J I^{ij} \bar{e}_j \tag{2.20}$$

  - o time reverse again to compute $J^T e_j = R_{n_i} \bar{z}_j$
- apply ILC algorithm (2.9), set $u_{j+1} = (I\epsilon W_f) u_j + \epsilon J^T W_e e_j$
- set $j = j + 1$ and go back to step 1 or stop if a suitable stopping criterion is met."

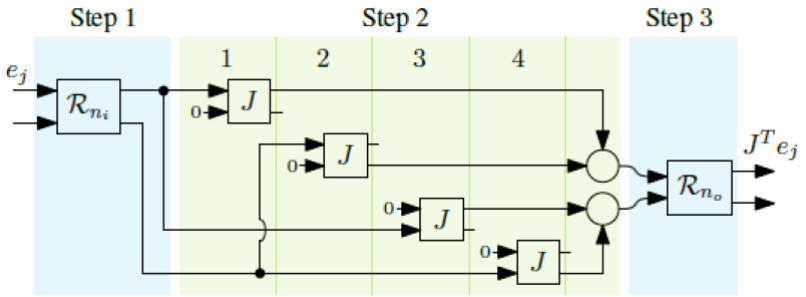

Figure 2.2: [18, Fig.2] Overview of the procedure for $n_i = n_o = 2$

Chapter 3

Robot NJ4 220 - 2.4

In this chapter is shown the first simplified SISO system of a 4^{th} link for the Comau robot *NJ4 220 – 2.4*.
All the schemes used for the simulation are reported in the appendix B while the MATLAB code is in A. Comau has provided two trajectories to work with: the first trajectory is a generic one while the second is a spot welding. In the following sections we talk about the steps done and the results obtained with three different ILC approaches, as illustrated in Chapter 2. We use a general procedure reported in [6] to develop them:

- we start using a Q-filter, as defined in 1, with an initial test frequency;
- when the result obtained is stable and good enough, the Q-filter is tuned to improve the convergence rate of the system at steady state.

3.1 PD-Type ILC

The first ILC tried is the simplest one: a PD ILC on a serial architecture (see Figure 4). As already said in the previous chapter we have to select a Q-filter to guarantee the convergence of the ILC algorithm due to model mismatch.
We choose a Butterworth lowpass filter of 3^{rd} order and for the cutoff frequency, we initially select it immediately below the resonance frequency of the robot, equal to 16Hz.
To select the optimal parameters for the PD-Type ILC we adopt as performance indicators the RMSE, 1-norm, 2-norm and ∞-norm of the errors at each iteration trying to minimize all or most of them. We consider both the error on the motor (em) and on the load (el) calculated as reported in Figure B.2.
We used the equation (2.1) and simulate the system cycling over k_p and k_d. We look for a first tuning where the system is AS and then we cycle near it, respectively in the range $[-0.5; 1.5]$ and $[-10; 20]$ with a step of 0.05 and 0.1. For each combination we reset the system and we apply the ILC for 50 iterations, stopping early only in the case that the

system becomes unstable or one of the performance indicators becomes larger than the initial one.

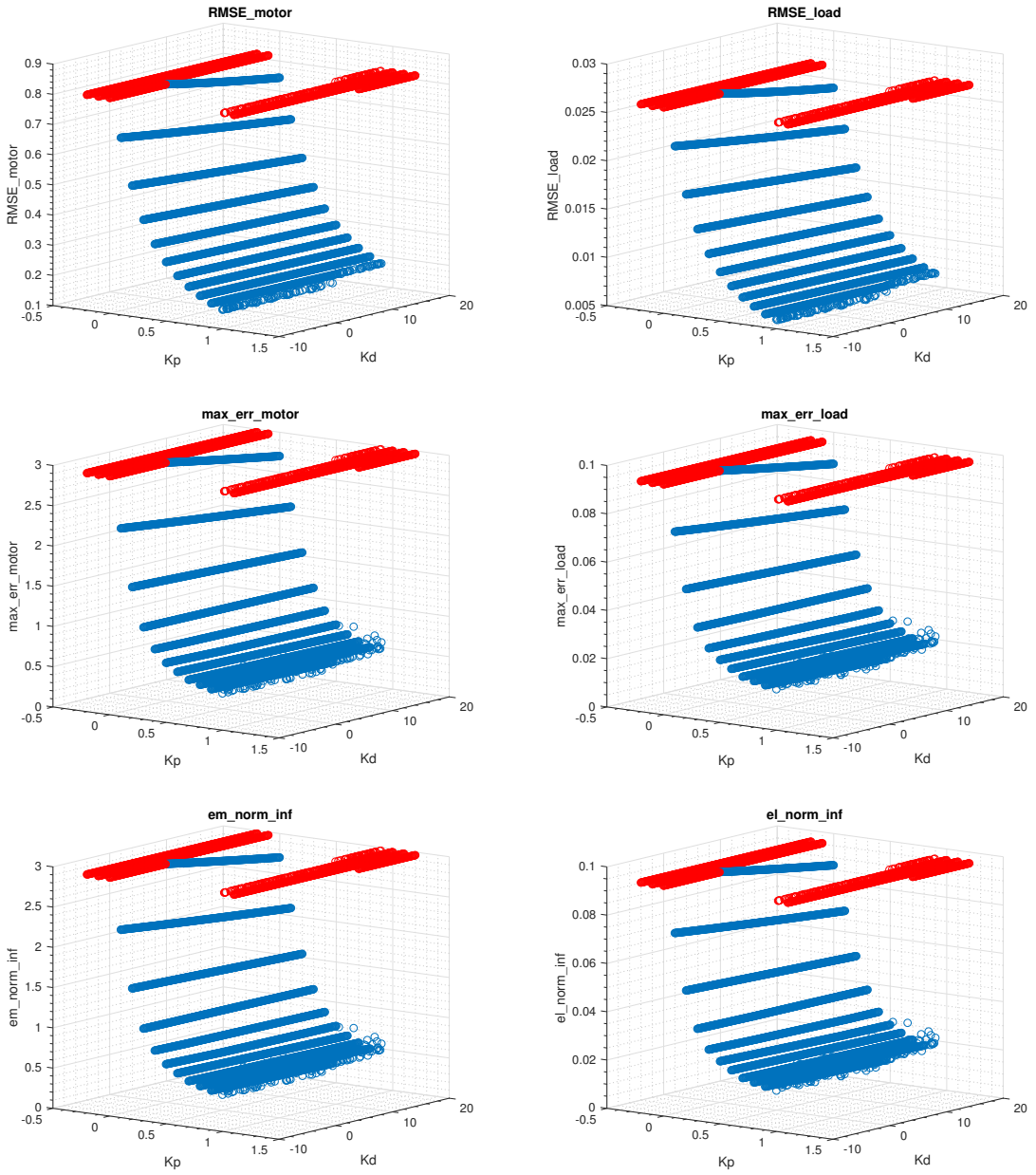

Figure 3.1: The blue points are stable for all the iterations instead the red ones are unstable. To be comparable, the red points are quoted to the initial iteration value.

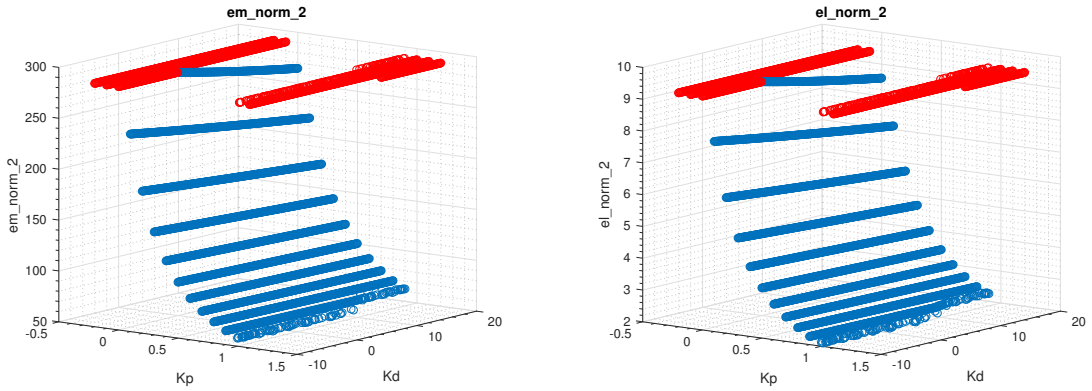

Figure 3.1: The blue points are stable for all the iterations instead the red ones are unstable. To be comparable, the red points are quoted to the initial iteration value. (cont.)

In Figure 3.1 are reported the plots of all the metrics, using the first trajectory. As can be seen increasing k_p and k_d the errors decrease, but when $k_p > 1$ the system becomes unstable. From these results we select the sub-optimal parameters $k_p = 1$ and $k_d = 5.8$ that minimize all the performance indicators.
Using these parameters, we simulate the system with the two trajectories for 50 iterations. Because the system is asymptotically stable after few trials the errors converge, so the plots are reported until the 20^{th} iteration.
For the first trajectory, reported in Figure 3.2, it can be seen that the errors are monotonically decreasing and converge after few iterations.

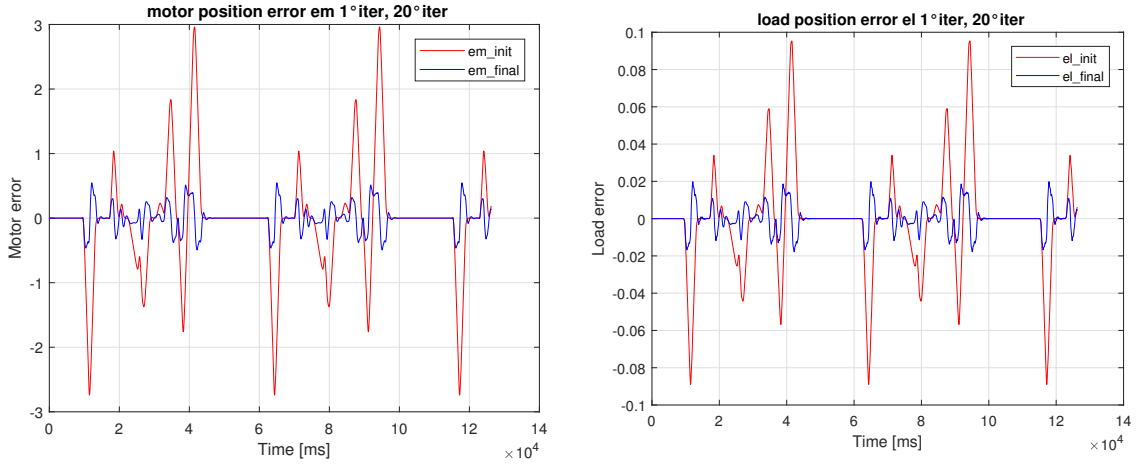

Figure 3.2: Errors and metrics for the first trajectory tested.

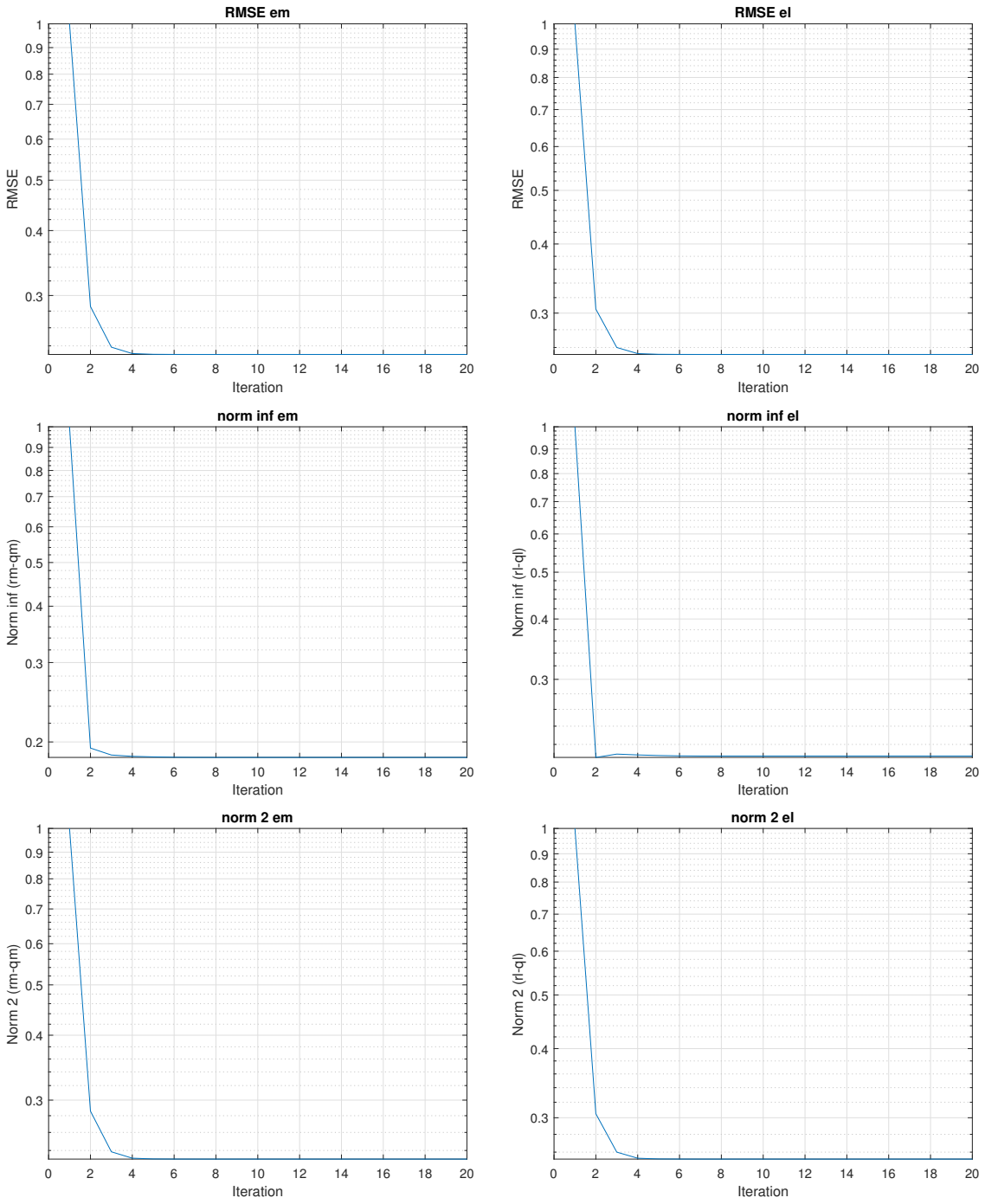

Figure 3.2: Errors and metrics for the first trajectory tested, the plots are normalized with respect to the max value and plot in semi-log. (cont.)

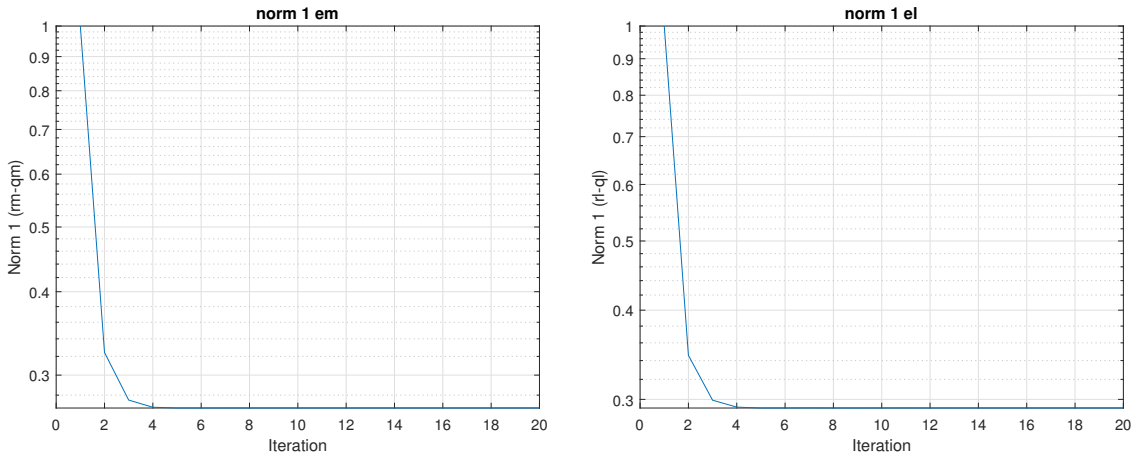

Figure 3.2: Errors and metrics for the first trajectory tested, the plots are normalized with respect to the max value and plot in semi-log. (cont.)

With the ILC, the system is generally more precise to follow the reference both on load and motor side but in some critical parts there are transitory conditions where the system does not follow perfectly the reference (see Figure 3.3 and Figure 3.4). The learning can be stopped after few iterations and is asymptotically stable.

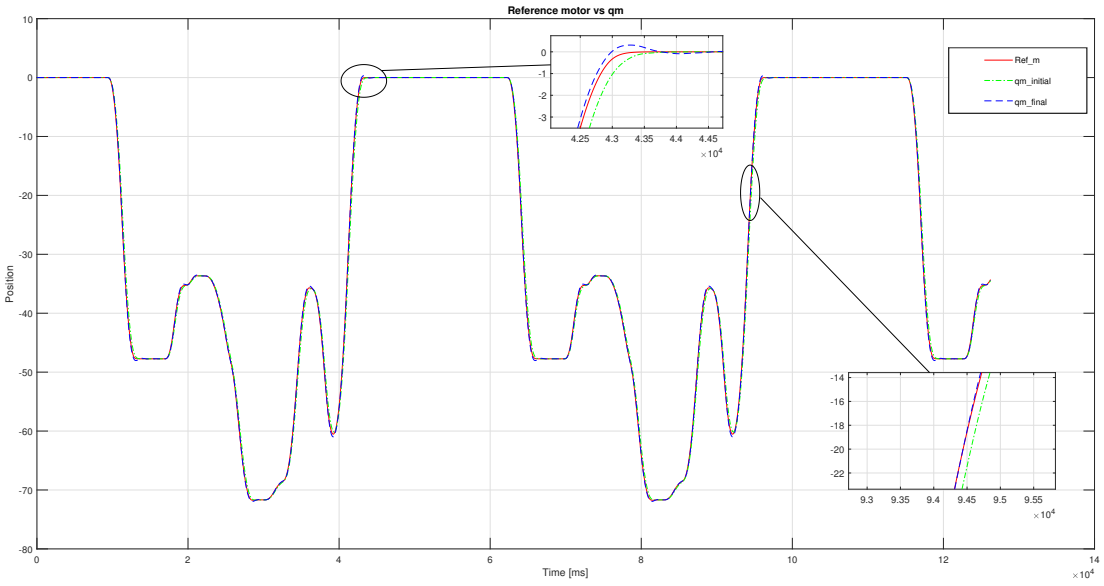

Figure 3.3: Comparison for motor position between the reference (red), the system without ILC (green) and the system plus ILC (blue) with a zoom during an important transient condition (on the top) and a less important transient (on the bottom).

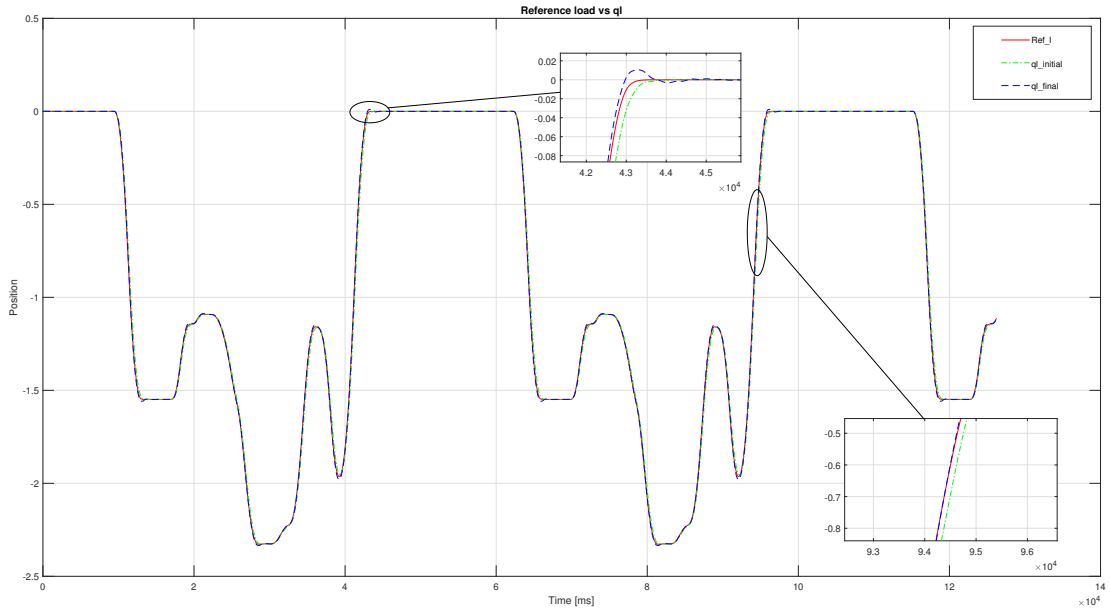

Figure 3.4: Comparison for load position between the reference (red), the system without ILC (green) and the system plus ILC (blue) with a zoom during an important transient condition (on the top) and a less important one (on the bottom).

These results are obtained using the resonance frequency of the robot, then the Q-filter is changed. We use other frequencies of the Q-filter $[1, 10, 16, 25, 40, 100]$Hz and simulate the system for 20 iterations (see Figure 3.5).

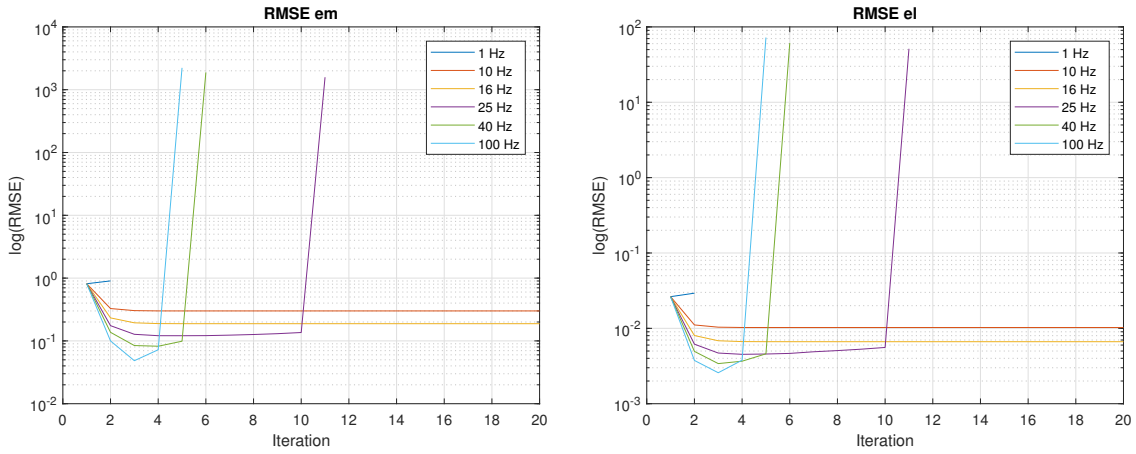

Figure 3.5: Metrics with different values of the cutoff frequency of the Q-filter.

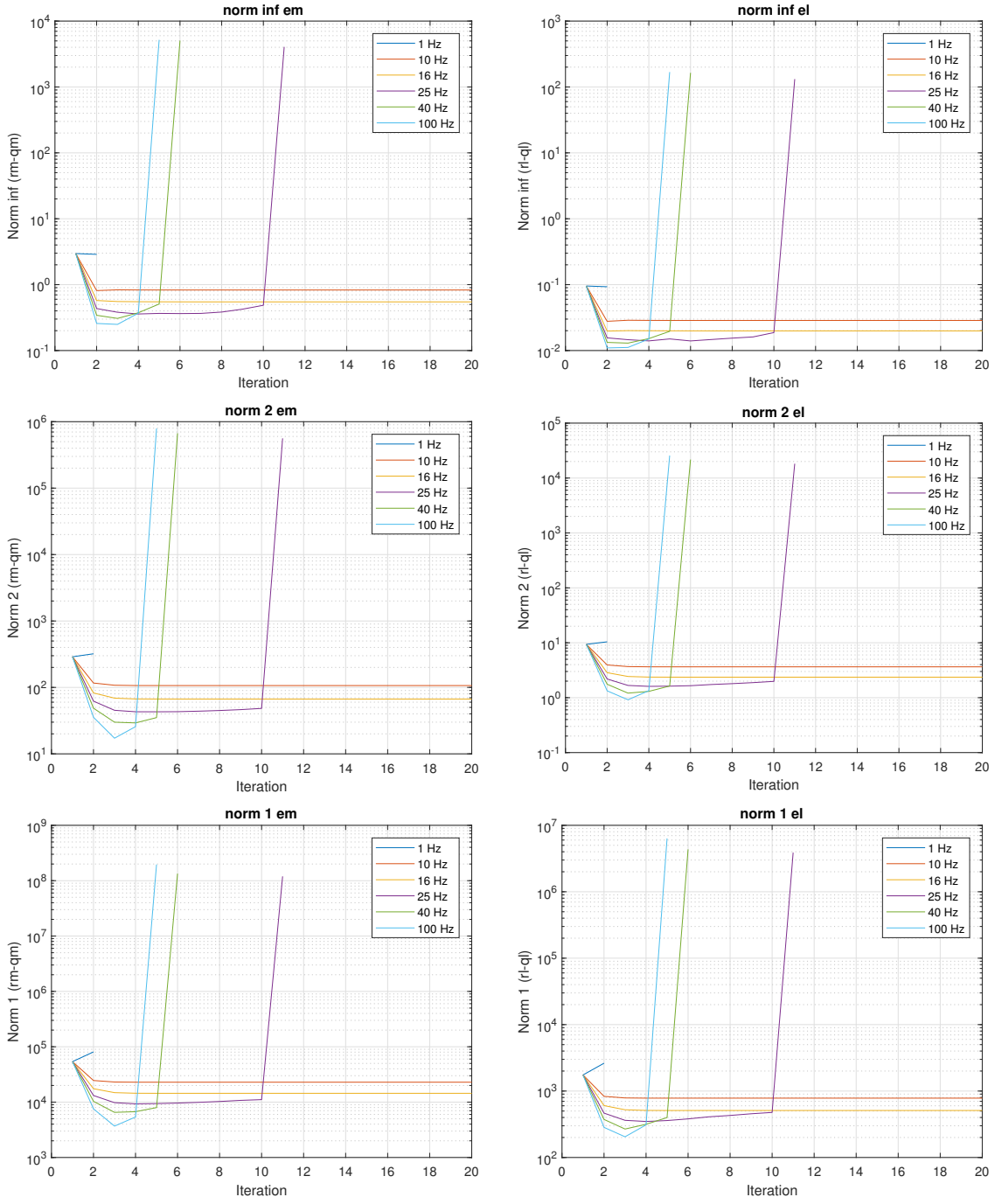

Figure 3.5: Metrics with different values of the cutoff frequency of the Q-filter. (cont.)

From Figure 3.5 it can be seen that the resonance frequency (16Hz) is the best value for the cutoff frequency. With a value too low the ILC performances decrease, while with a frequency too high it induces vibrations on the system that lead to instability.
In a similar way we do for the second trajectory. The parameters chosen are the best solution for the first trajectory: used also for the second one we notice that the PD-Type ILC is less effective; the error is smaller than the initial one and is not monotonically decreasing. From the 2^{nd} to the 6^{th} iteration the metrics increase and then they become stable (see Figure 3.6).

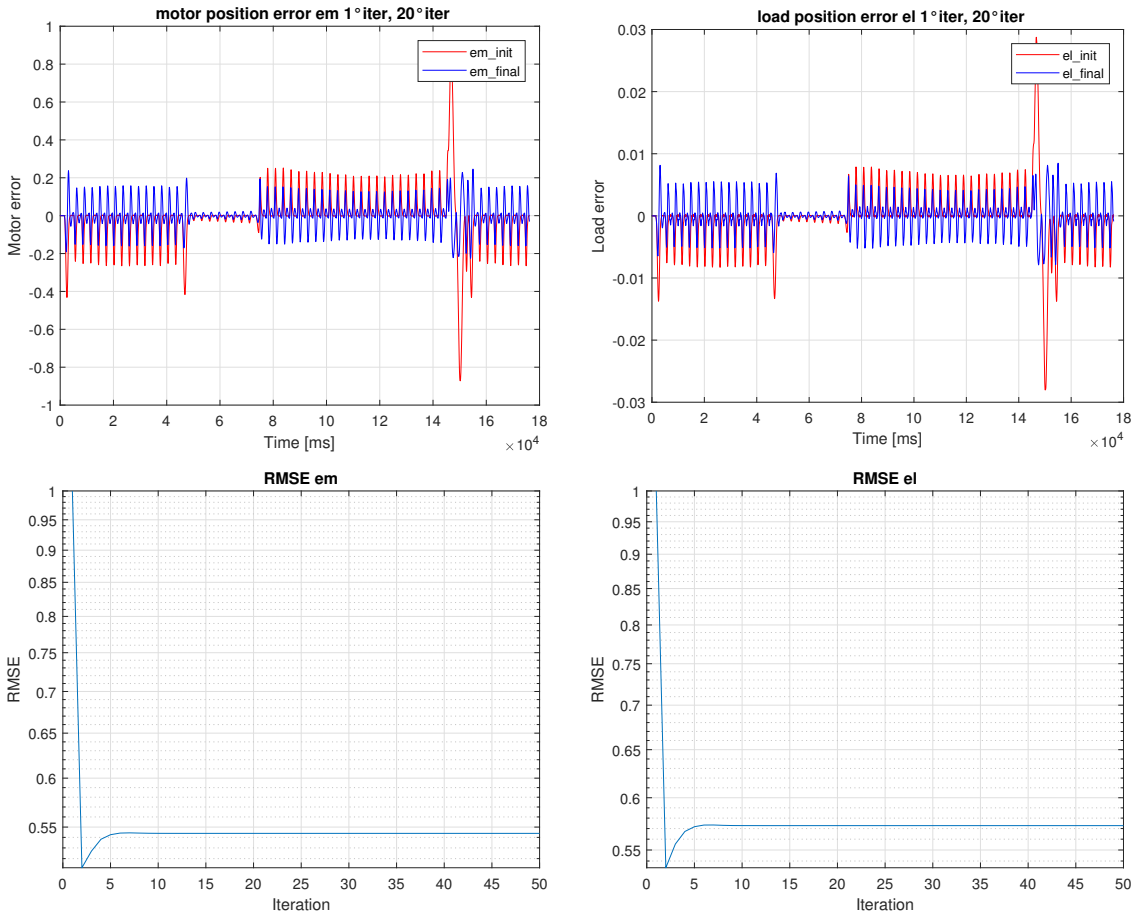

Figure 3.6: Errors and metrics for the second trajectory tested, the plots are normalized with respect to the max value.

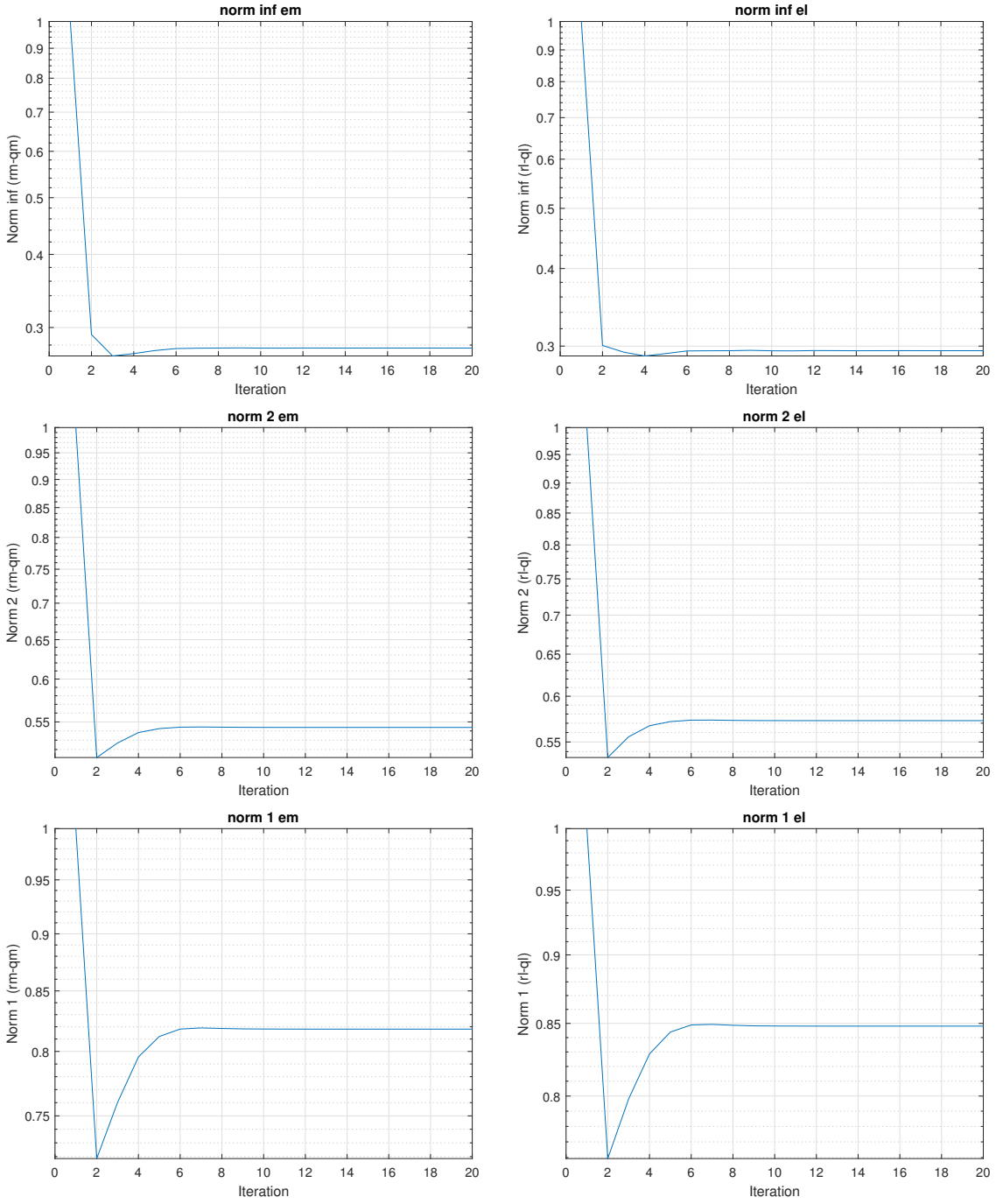

Figure 3.6: Errors and metrics for the second trajectory tested, the plots are normalized with respect to the max value. (cont.)

With the ILC, the system is generally more precise to follow the reference both on load and motor side (see Figure 3.7 and Figure 3.8), in some points the result is slightly worse.

This is probably due to the Q-filter that cutoff the high frequency and the system fails to follow rapidly the reference.

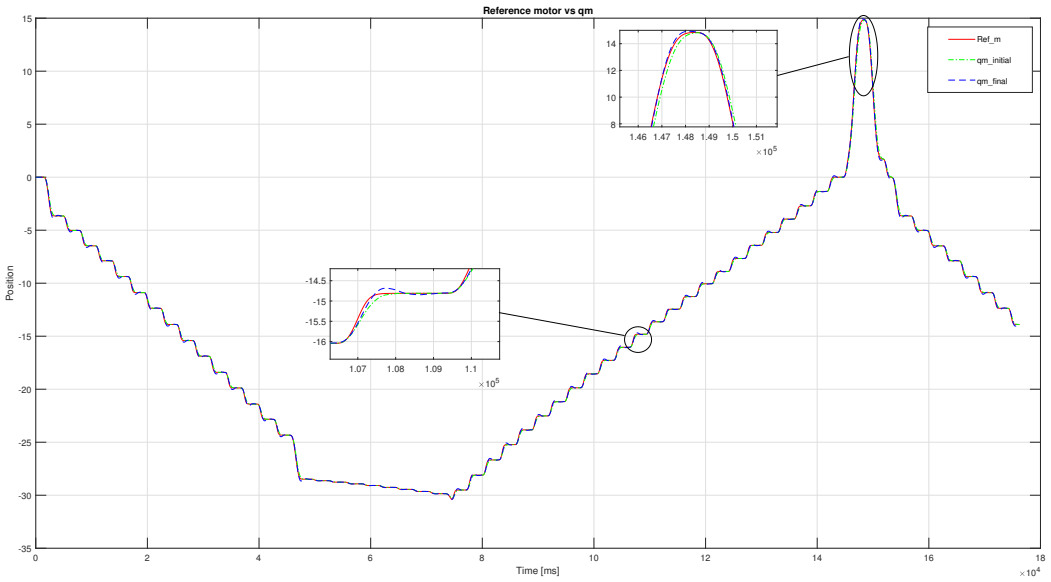

Figure 3.7: Comparison for motor position between the reference (red), the system without ILC (green) and the system plus ILC (blue) with a zoom during an important transient condition (on the bottom) and a less important one (on the top).

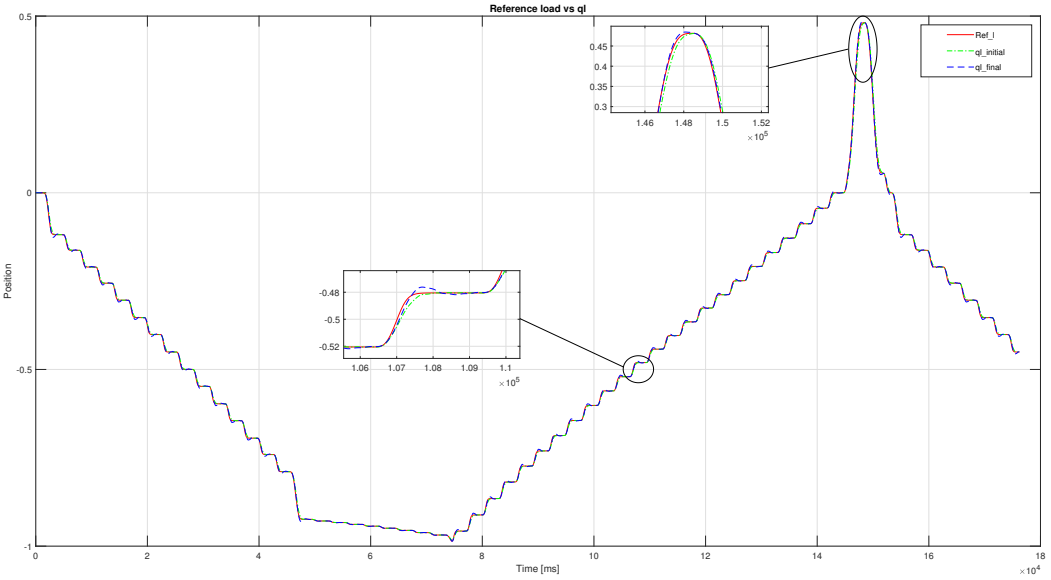

Figure 3.8: Comparison for load position between the reference (red), the system without ILC (green) and the system plus ILC (blue) with a zoom during an important transient condition (on the bottom) and a less important transient (on the top).

Trying to increase the frequency of the Q-filter we have the same problem of the first trajectory: the system becomes unstable (see Figure 3.9), so we choose as cutoff frequency for Q-filter 16Hz that is the best one.

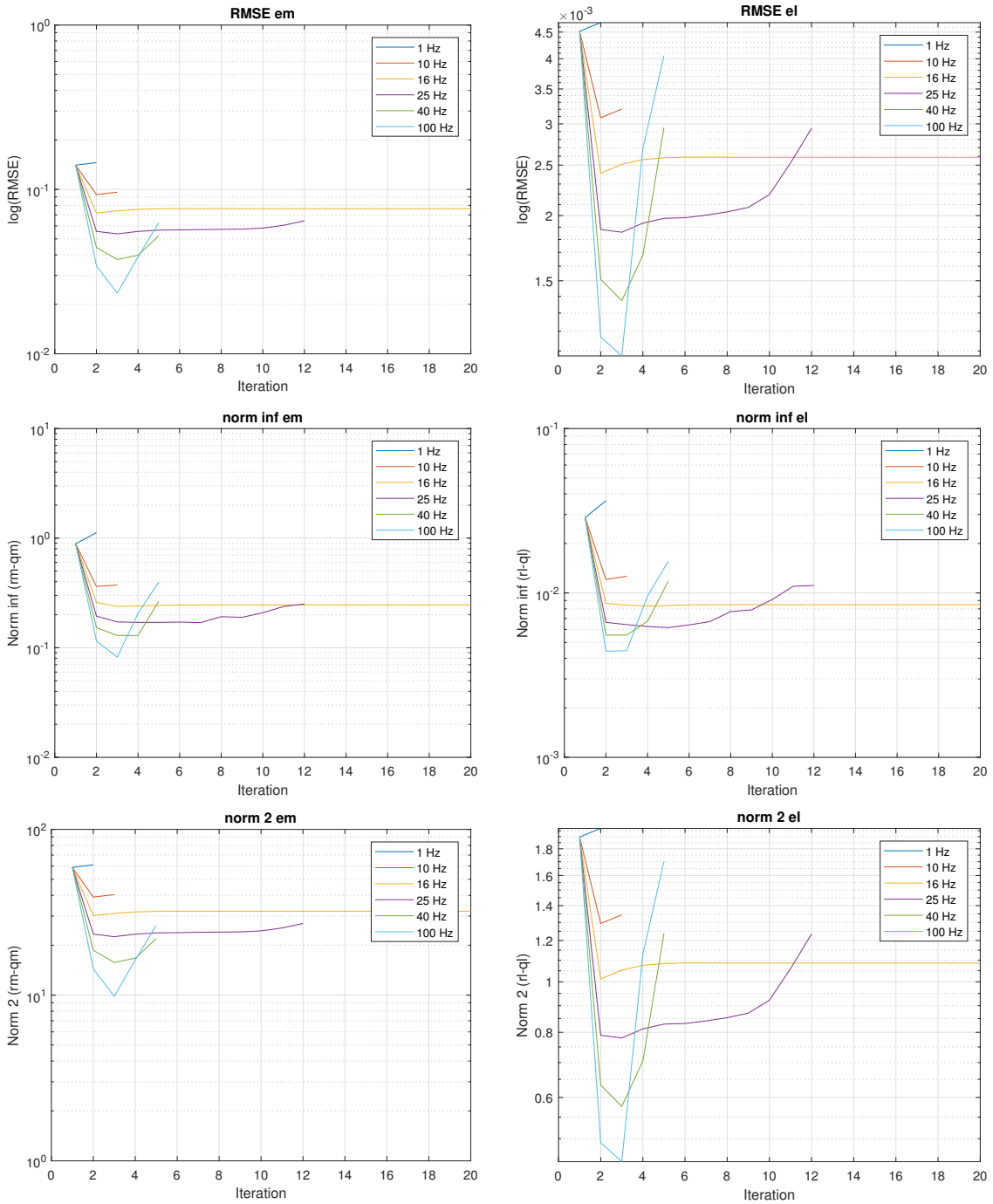

Figure 3.9: Metrics with different values of the cutoff frequency of the Q-filter.

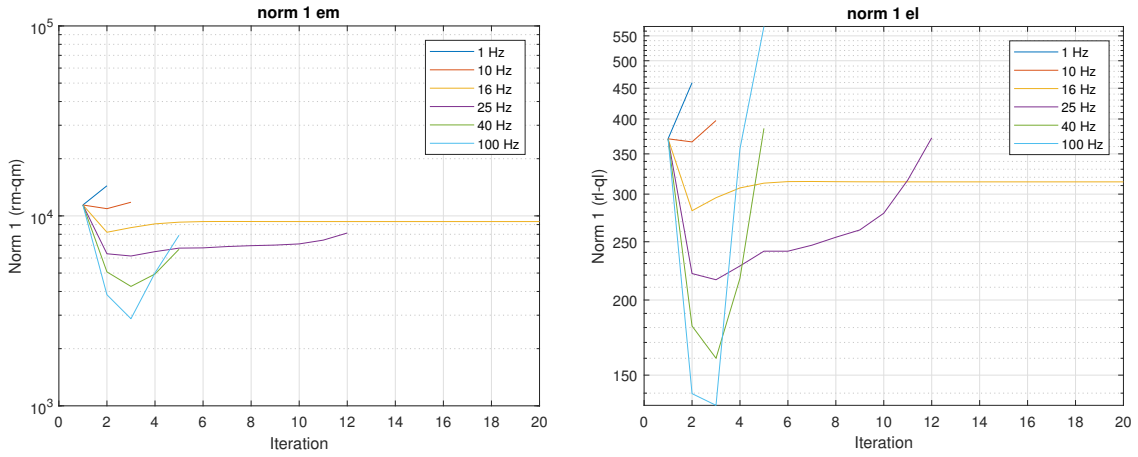

Figure 3.9: Metrics with different values of the cutoff frequency of the Q-filter. (cont.)

3.2 Plant Inversion ILC

For inverse plant approach is very important the accuracy of the model as remarked in Chapter 2.2. Because we are applying the ILC command to the reference, the plant seen by the ILC is the overall system. The transfer function from the reference input to the motor output is nonminimum phase then, to obtain a stable inverse, we combine some pole/zero cancellation and phase cancellation techniques [37].
We report the fundamental steps to obtain a stable inverse with the assumptions that the starting model is a good one.
We rewrite the system in the form

$$G_{closed}(z^{-1}) = \frac{z^{-d}B_c(z^{-1})}{A_c(z^{-1})} \tag{3.1}$$

where z^{-d} represents a d-step delay normally caused by a delay in the plant. We factorize $B_c(z^{-1})$ into two parts and write

$$B_c(z^{-1}) = B_c^a(z^{-1})B_c^u(z^{-1})$$

where $B_c^a(z^{-l})$ includes zeros of the closed-loop system which are sufficiently inside the unit circle, and $B_c^u(z^{-l})$ includes those outside or close to the unit circle.
The tracking controller which cancels all the closed-loop poles and zeros from $B_c^a(z^{-1})$ is

$$r(k) = \frac{A_c(z^{-1})}{B_c^a(z^{-1})B_c^u(1)}y_d^*(k+d) \tag{3.2}$$

$$y_d^*(k+d) = \frac{B_c^u(z)}{B_c^u(1)}y_d(k) \tag{3.3}$$

where $y_d^*(k)$ is related to the desired trajectory $y_d(k)$ and $B_c^u(z^{-1})$ in the denominator is to scale the steady state gain to the reciprocal of that of the closed-loop transfer function given by equation (3.1).
The final result is obtained substituting (3.3) in (3.2)

$$r(k) = \frac{A_c(z^{-1})B_c^{u^*}(z^{-1})}{B_c^a(z^{-1})B_c^u(1)} y_d^*(k+d+s) \tag{3.4}$$

Where the $(d+s)$-step ahead is the desired output. By using this transfer function, the phase shift between $y_d(k)$ and $y(k)$ is zero for all frequencies.
Equation (3.4) is called the zero-phase error tracking controller (ZPETC) and is a good approximation of the inverse needed. We calculate the inverse of the plant (P^{-1}) and then multiplying for the plant (P); if the inverse is a good one the bode of PP^{-1} should give a zero magnitude and phase but, due to approximations, this is true until a certain low frequency, as can be seen in Figure 3.10.

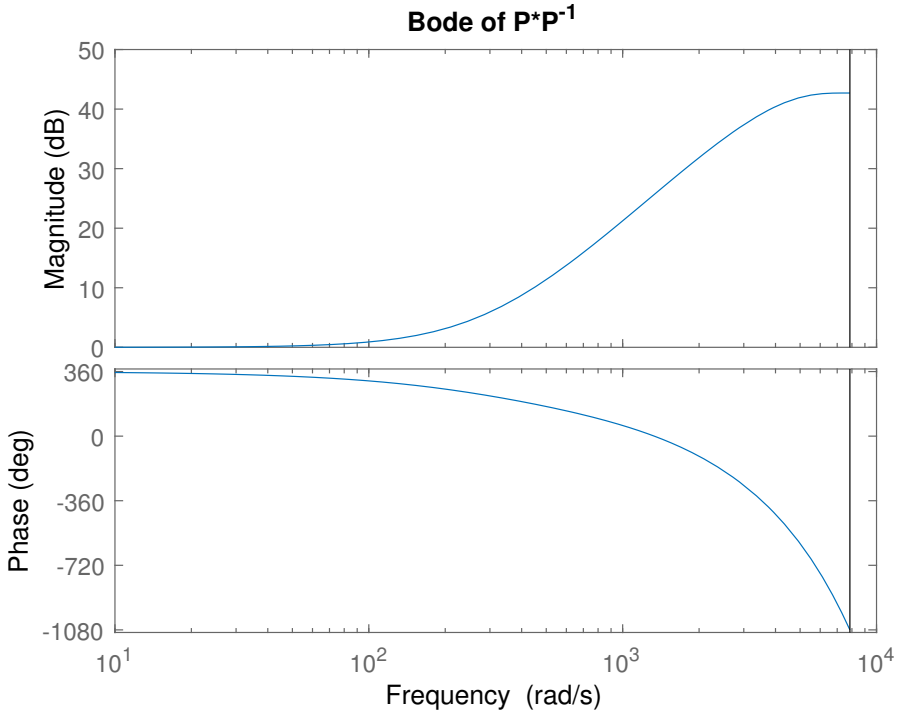

Figure 3.10: Magnitude and phase Bode diagram of PP^{-1}.

Due to model uncertainty we need to add a Q-filter to ensure that the system is stable. The results obtained by the inverse approach with a Q-filter at same frequency of the PD-Type are worse than PD-Type in particular in some critical points. To improve the quality

of the results we increase the value of the cutoff frequency to 100Hz that, as can be seen in Figure 3.18, Figure 3.19 and explained later on, is the best one. For the first trajectory we report in Figure 3.11 the errors and metrics and in Figure 3.12 and Figure 3.13 the system position.

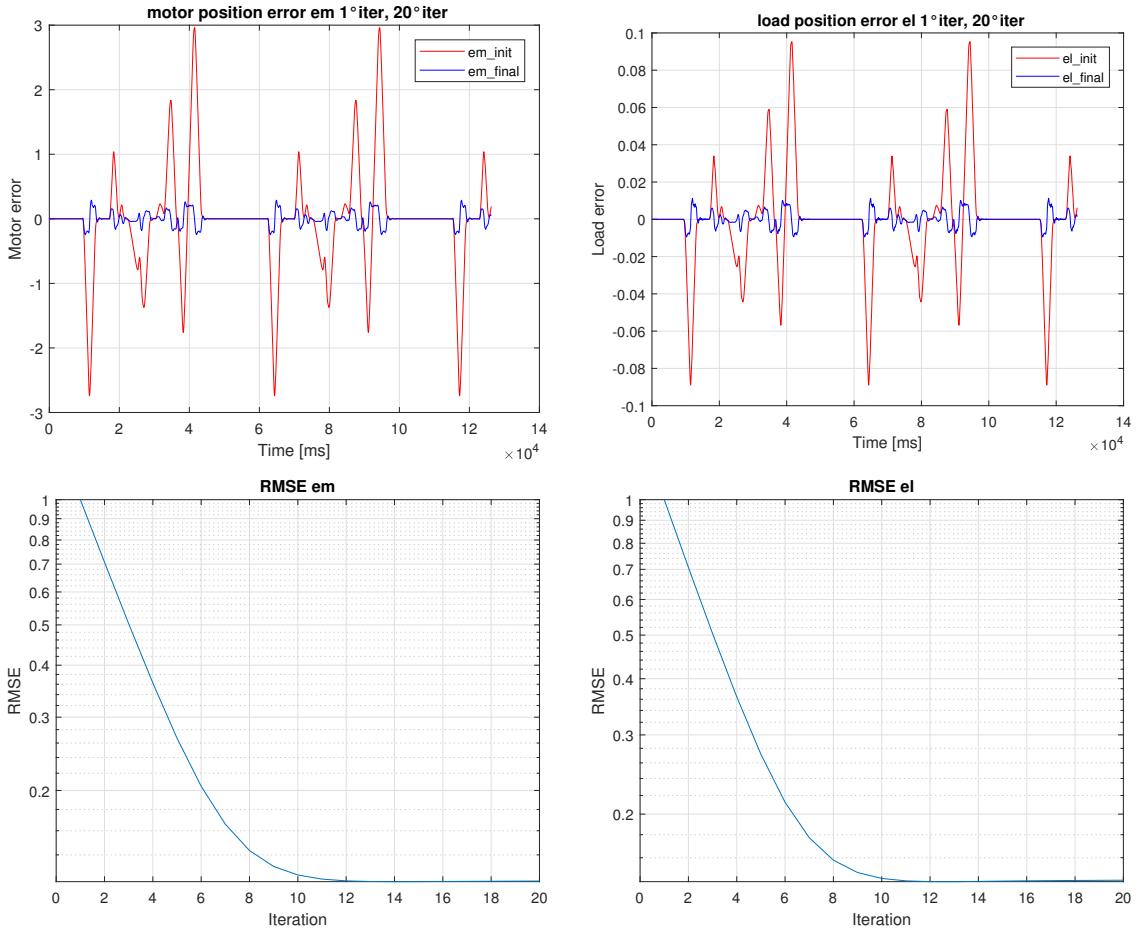

Figure 3.11: Errors and metrics using inverse plant approach on the first trajectory.

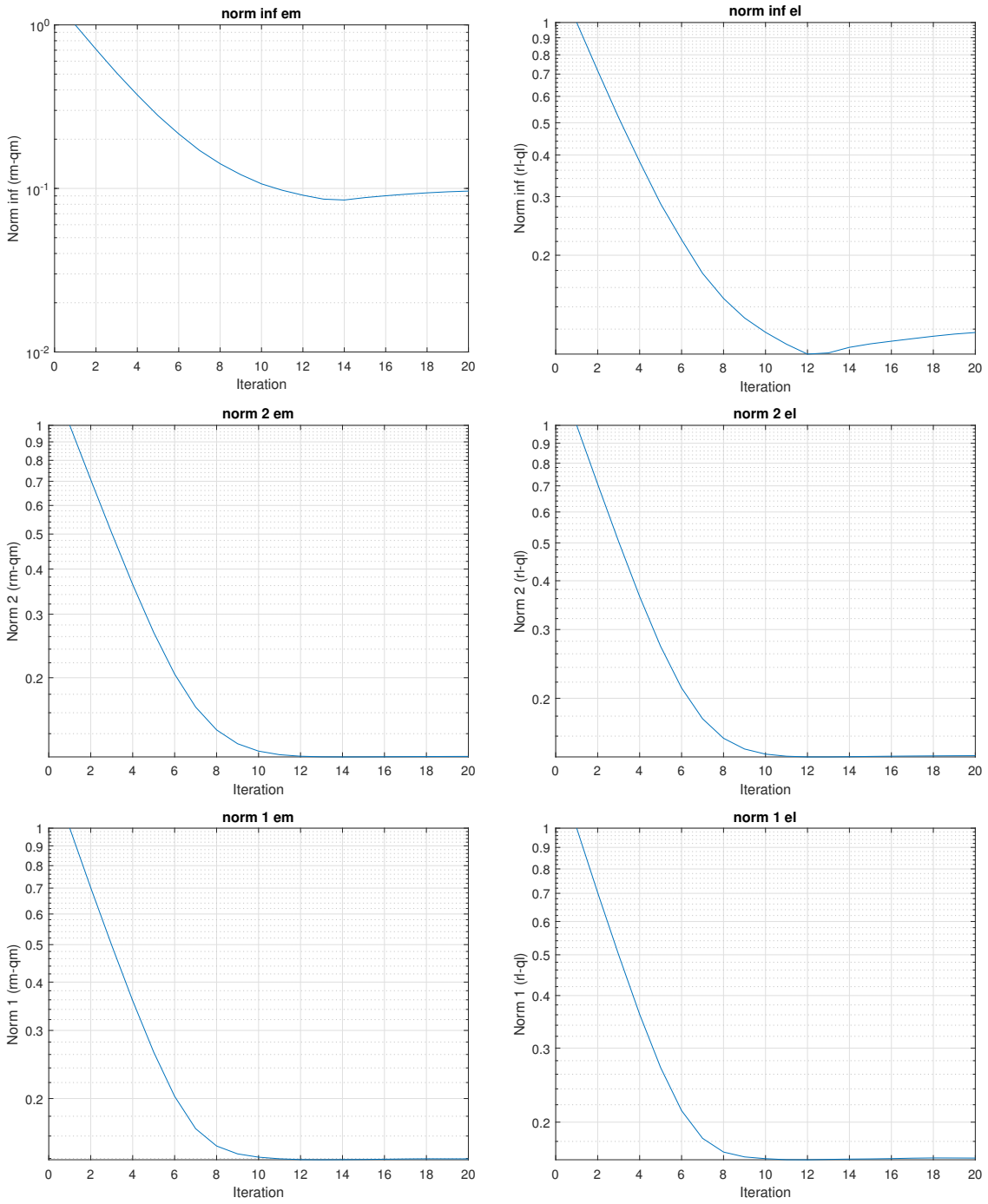

Figure 3.11: Errors and metrics using inverse plant approach on the first trajectory (cont.)

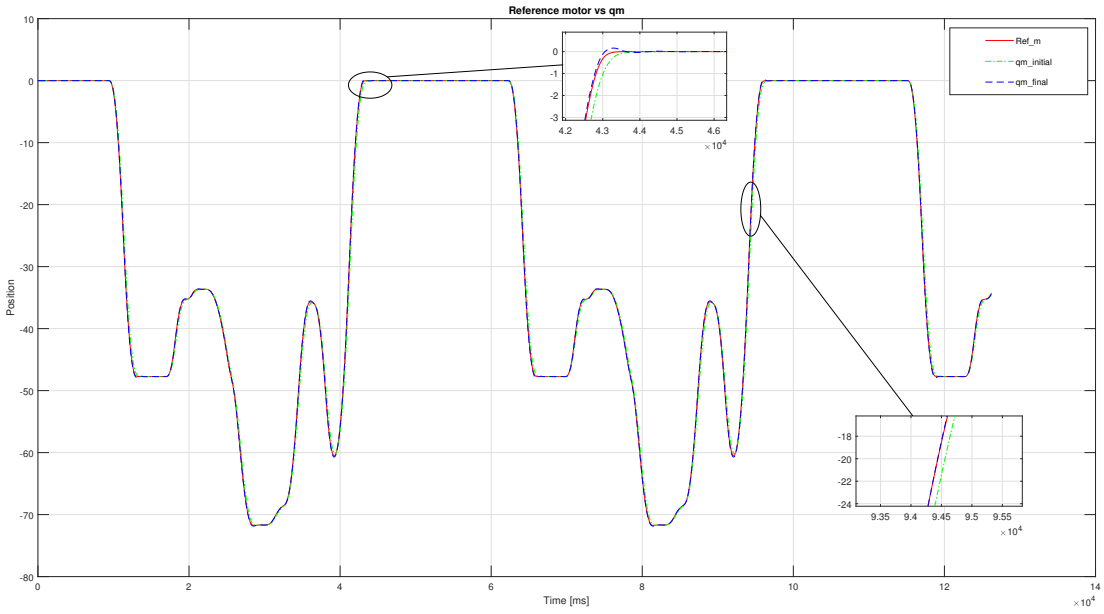

Figure 3.12: Comparison for motor position between the reference (red), the system without ILC (green) and the system plus ILC (blue) with a zoom during an important transient condition (on the top) and a less important transient (on the bottom).

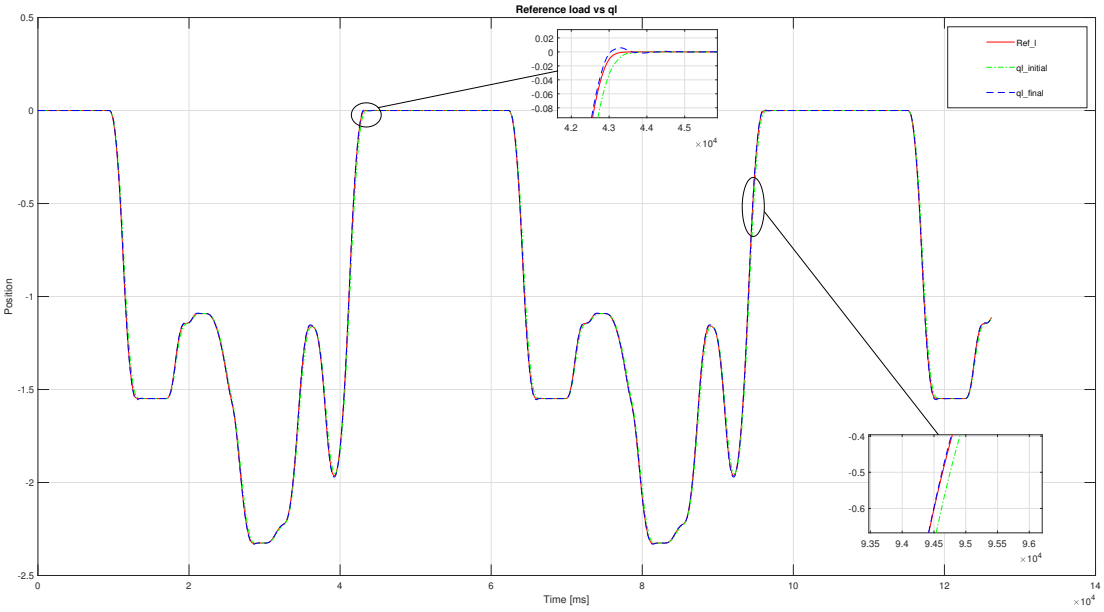

Figure 3.13: Comparison for load position between the reference (red), the system without ILC (green) and the system plus ILC (blue) with a zoom during an important transient condition (on the top) and a less important transient (on the bottom).

Similarly to previous trajectory, for the second one the results are reported in Figure 3.14, Figure 3.15 and Figure 3.16

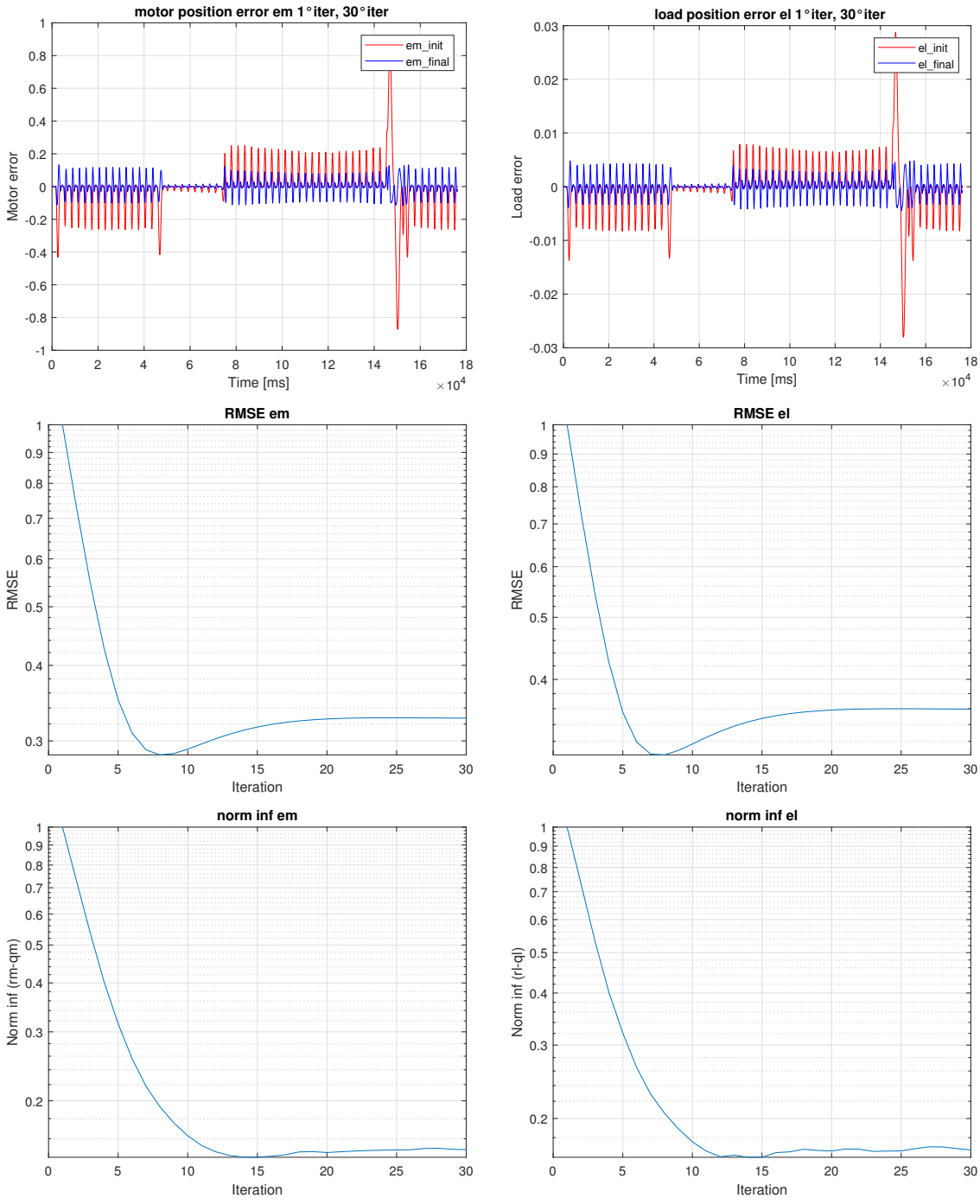

Figure 3.14: Errors and metrics using inverse plant approach on the second trajectory.

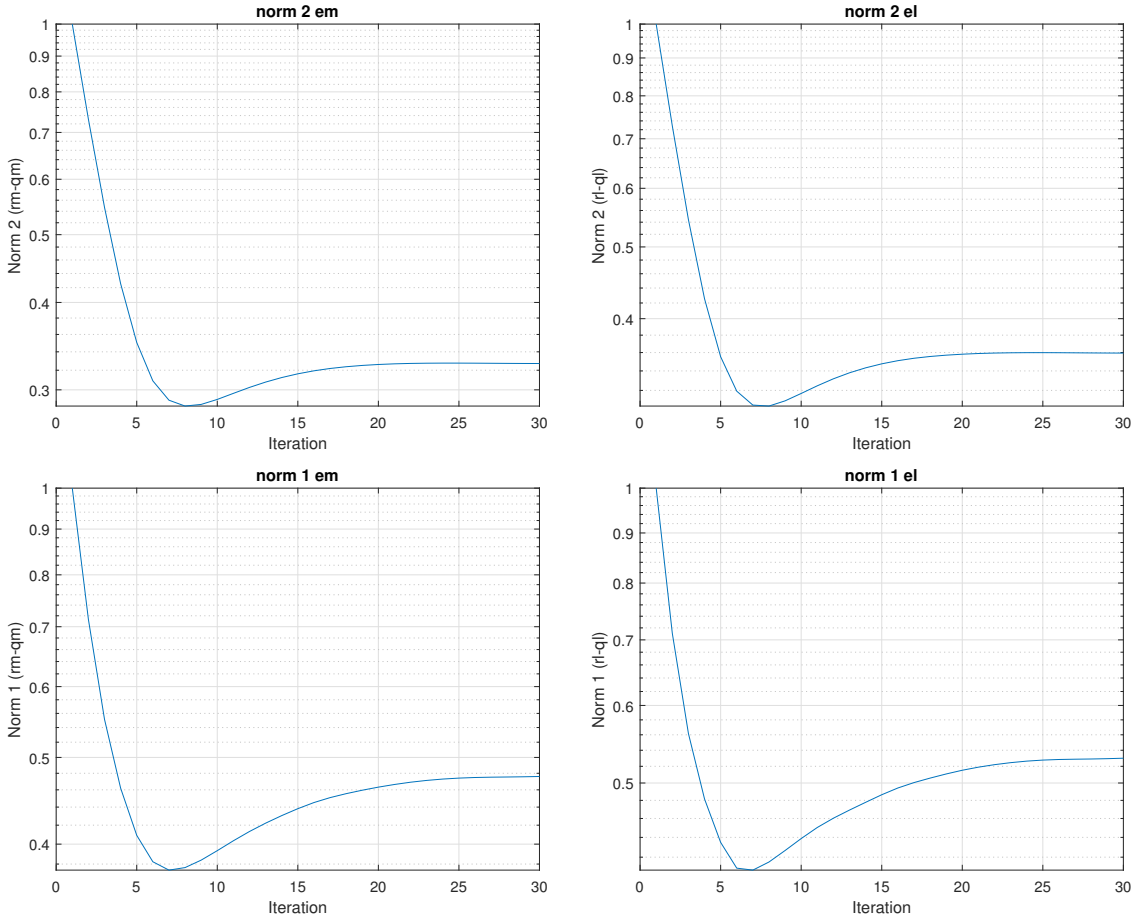

Figure 3.14: Errors and metrics using inverse plant approach on the second trajectory. (cont.)

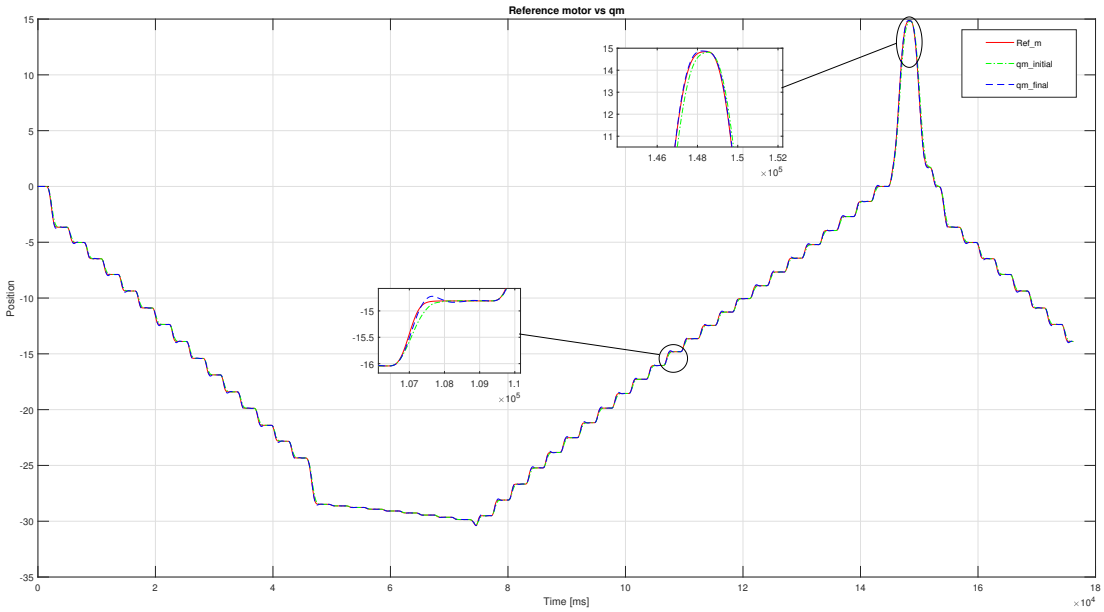

Figure 3.15: Comparison for motor position between the reference (red), the system without ILC (green) and the system plus ILC (blue) with a zoom during an important transient condition (on the bottom) and a less important one (on the top).

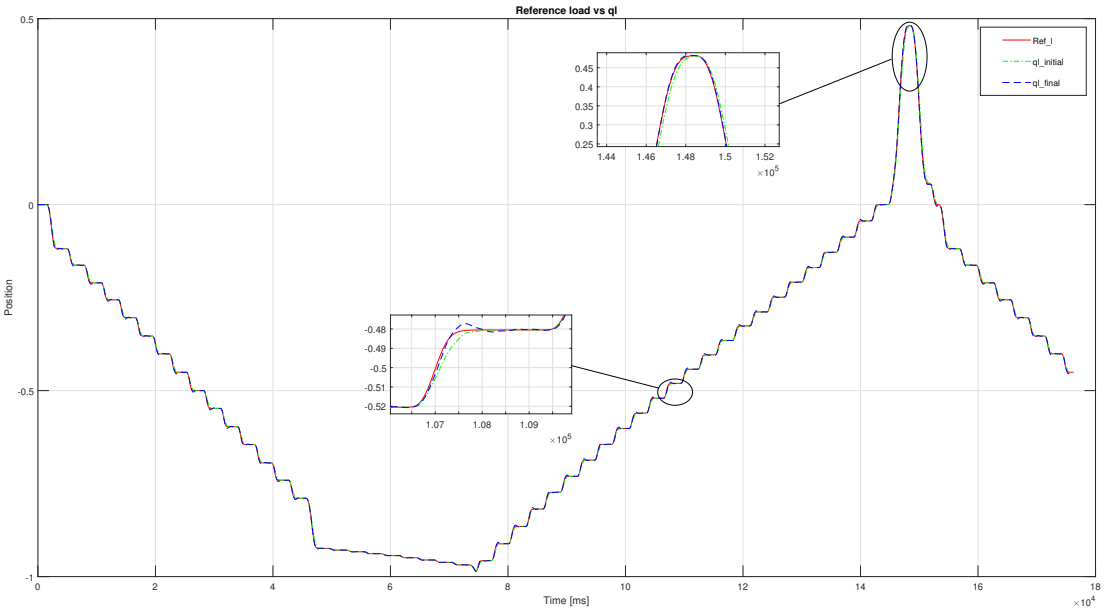

Figure 3.16: Comparison for load position between the reference (red), the system without ILC (green) and the system plus ILC (blue) with a zoom during an important transient condition (on the bottom) and a less important one (on the top).

For the Q-filter can be done similar considerations of the PD-Type: with a value too low

of frequency the ILC cannot correctly follow the error and apply a valid correction, instead with a value too high induces vibrations on the system that lead to instability. To choose the best frequency we select the global minimization of all the metrics in Figure 3.17 and Figure 3.18 and finally select the cutoff frequency at 100Hz.

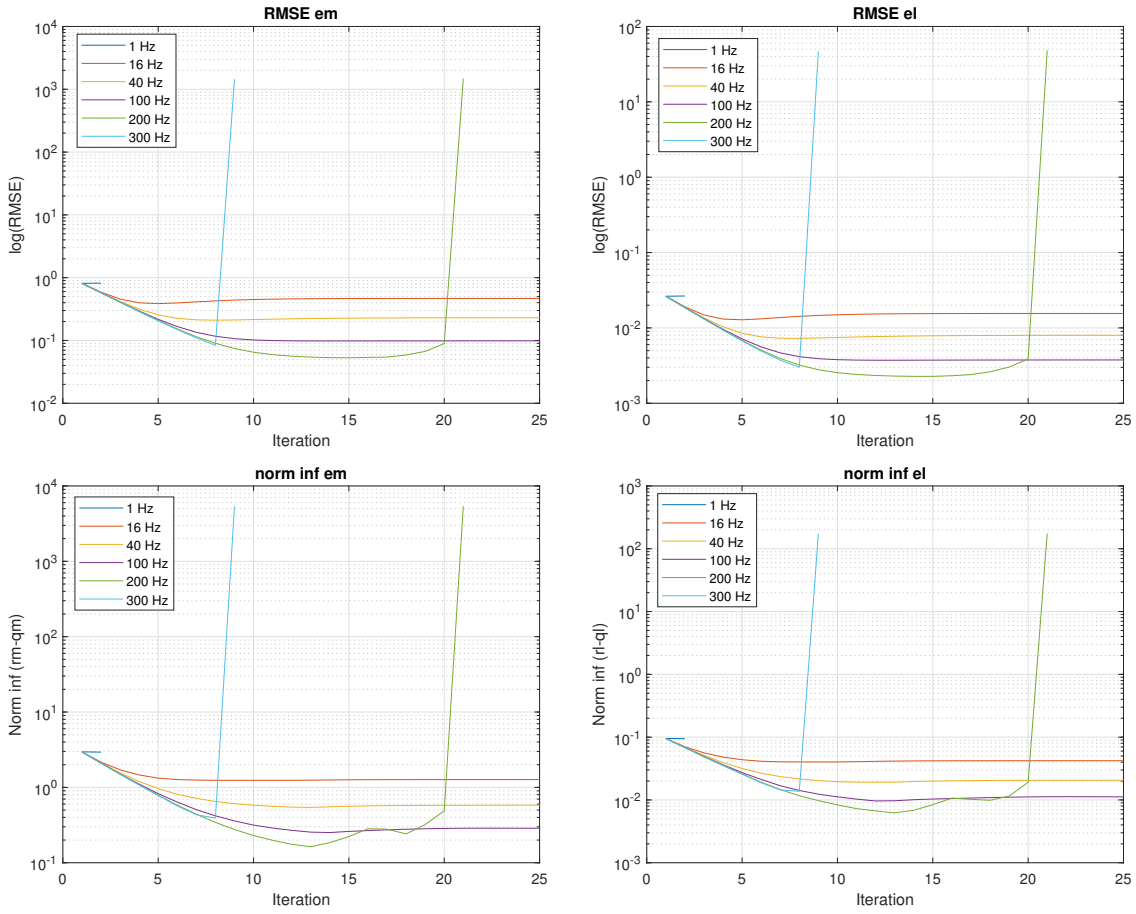

Figure 3.17: For the first trajectory, metrics with different values of the cutoff frequency of the Q-filter.

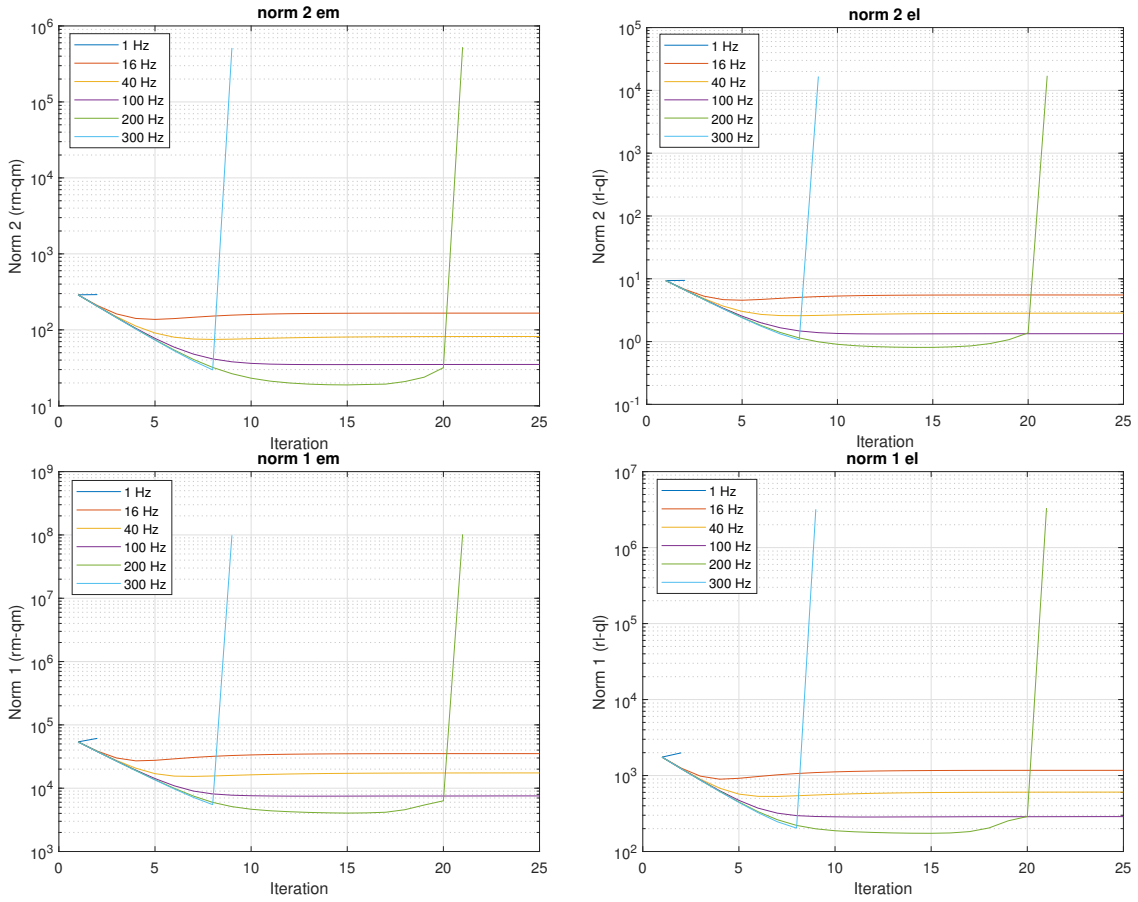

Figure 3.17: For the first trajectory, metrics with different values of the cutoff frequency of the Q-filter. (cont.)

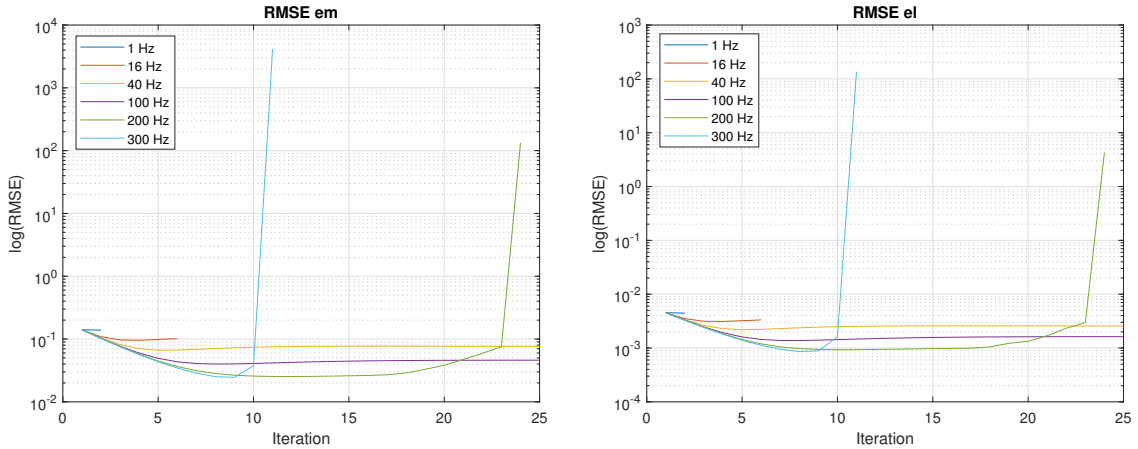

Figure 3.18: For the second trajectory, metrics with different values of the cutoff frequency of the Q-filter.

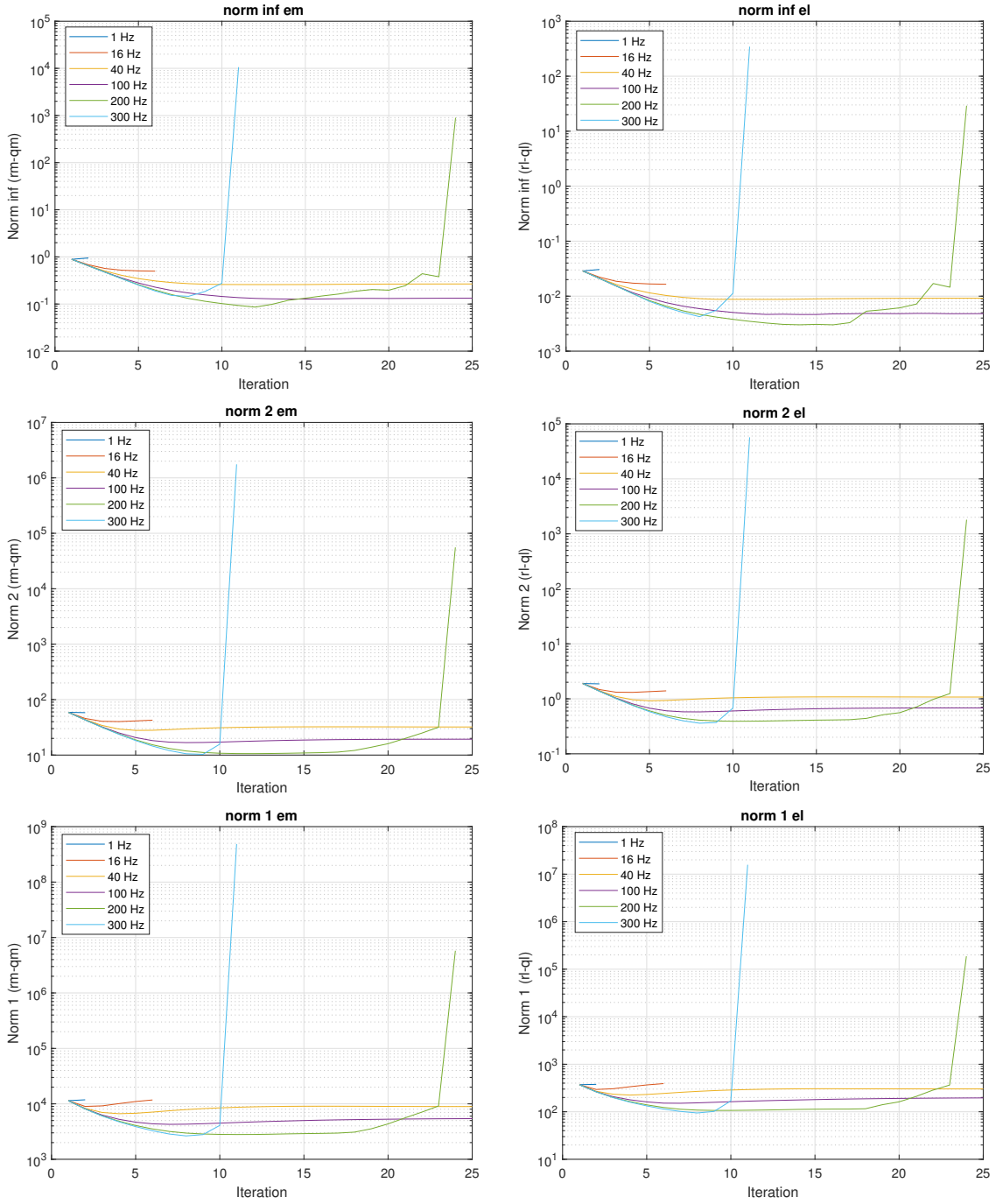

Figure 3.18: For the second trajectory, metrics with different values of the cutoff frequency of the Q-filter. (cont.)

3.3 Data Driven ILC

The Data Driven ILC does not require directly the computation of L filter, instead it uses the Markov parameters to evaluate the command to apply to the reference.
The Markov parameters h are computed using a reduced state space representation of the system with the following definition

$$\mathbf{h}(n) = \begin{cases} D, & \text{if } n = 0 \\ CA^{n-1}B, & \text{if } n > 0 \end{cases}$$

This is slightly different from the approach adopted in [18] where the authors experimentally identifies the J matrix and so the h parameters. We use the theoretical definition and a state-reduced mathematical model obtained from the linear analysis of the Simulink model. The ILC command is computed as reported in the previous chapter. We use $W_e = I$ (I = identity matrix) and $\epsilon = 0.8$ as weight for the command evaluation and a Q-filter, with a cutoff frequency of 16Hz, to guarantee the stability of the system.
For the first trajectory we report in Figure 3.19 the errors and metrics and in Figure 3.20 and Figure 3.21 the system position.

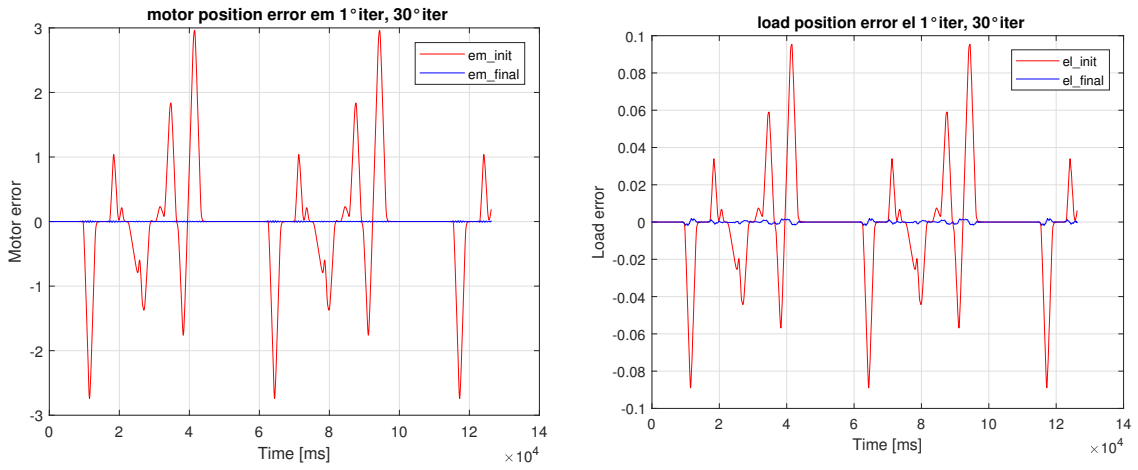

Figure 3.19: Errors and metrics using Data Driven approach on the first trajectory.

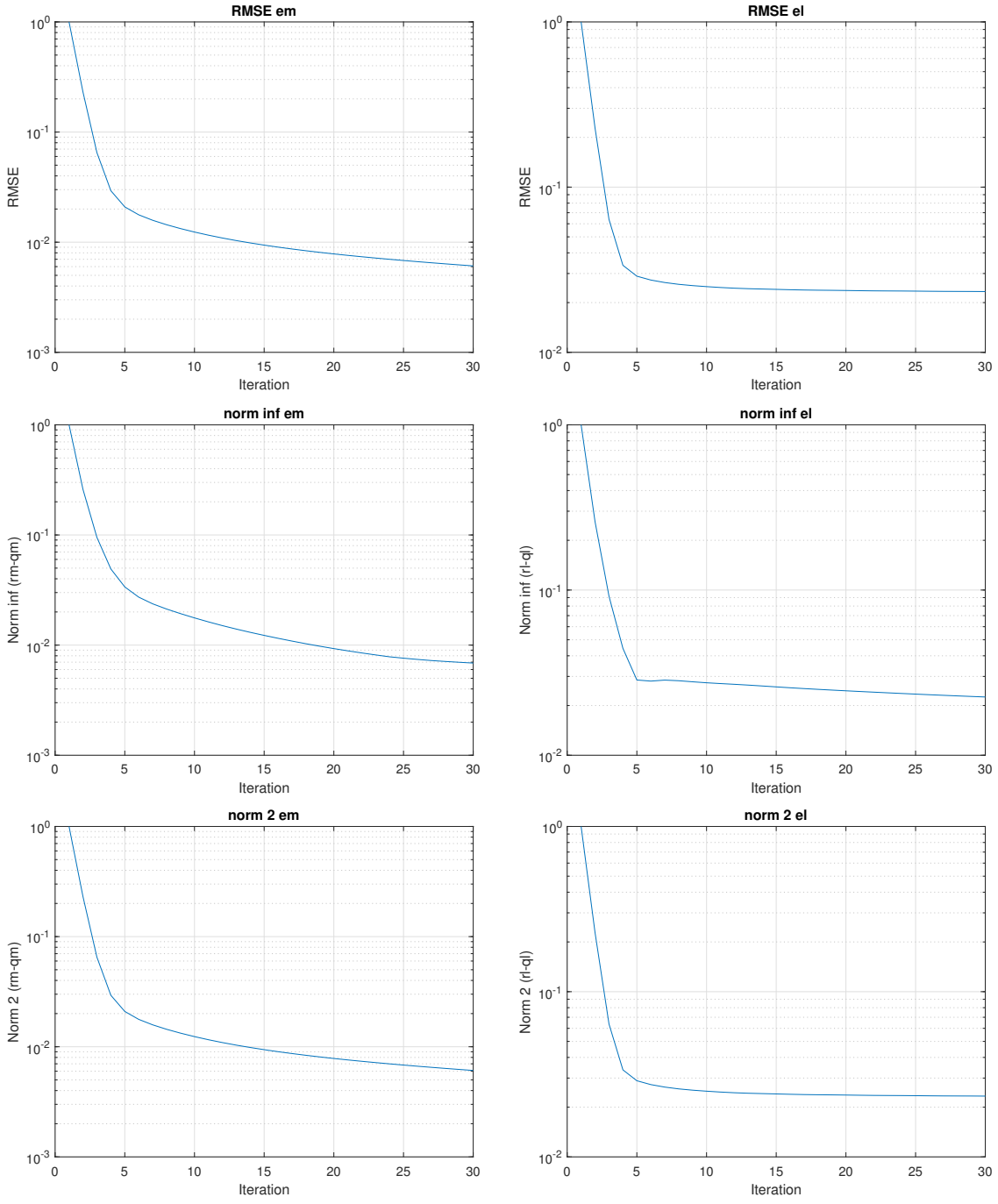

Figure 3.19: Errors and metrics using Data Driven approach on the first trajectory. (cont.)

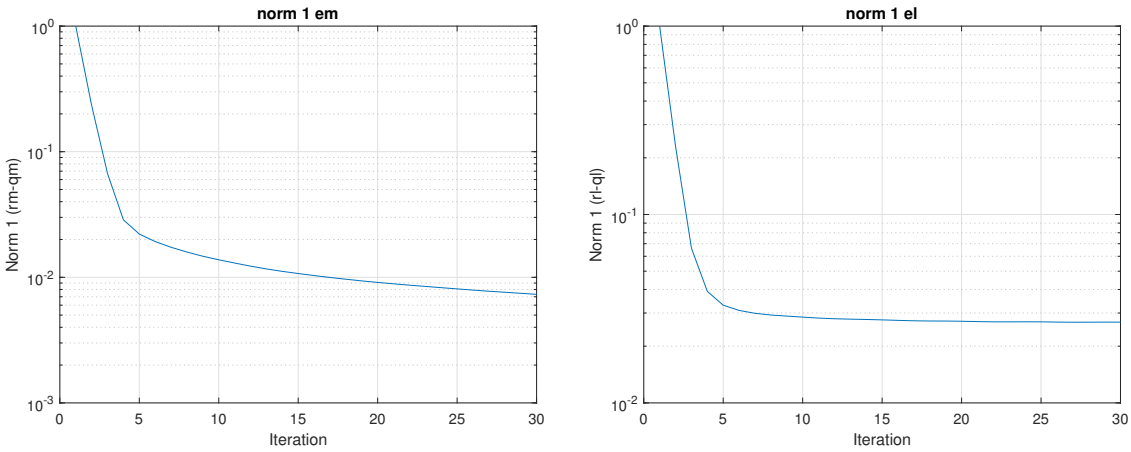

Figure 3.19: Errors and metrics using Data Driven approach on the first trajectory (cont.)

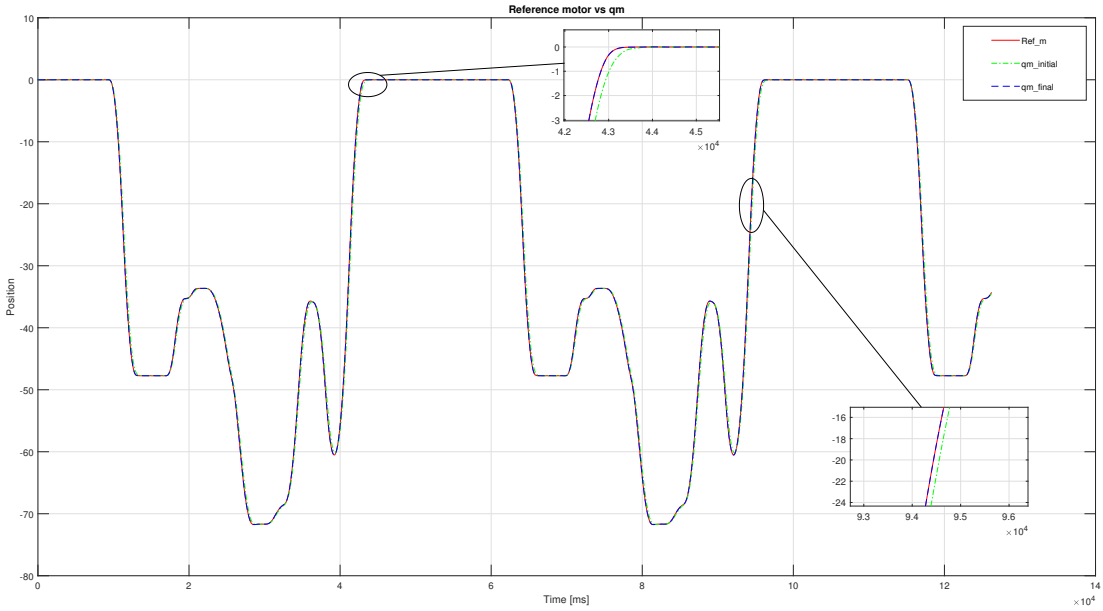

Figure 3.20: Comparison for motor position between the reference (red), the system without ILC (green) and the system plus ILC (blue) with a zoom during an important transient condition (on the top) and a less important one (on the bottom).

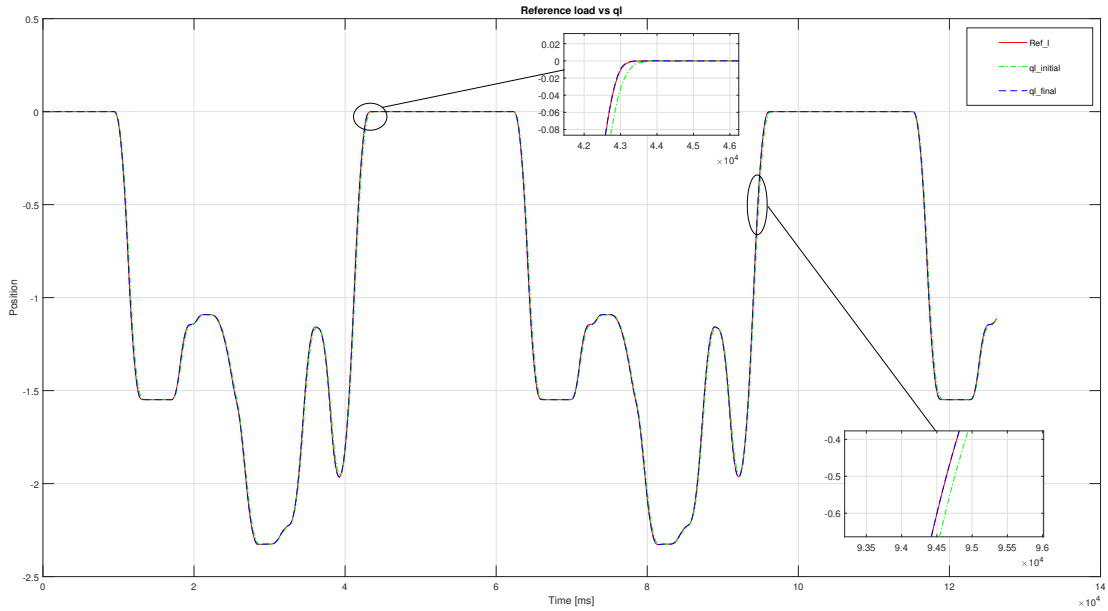

Figure 3.21: Comparison for load position between the reference (red), the system without ILC (green) and the system plus ILC (blue) with a zoom during an important transient condition (on the top) and a less important one (on the bottom).

Similarly to previous trajectory, for the second one we have Figure 3.22, Figure 3.23 and Figure 3.24.

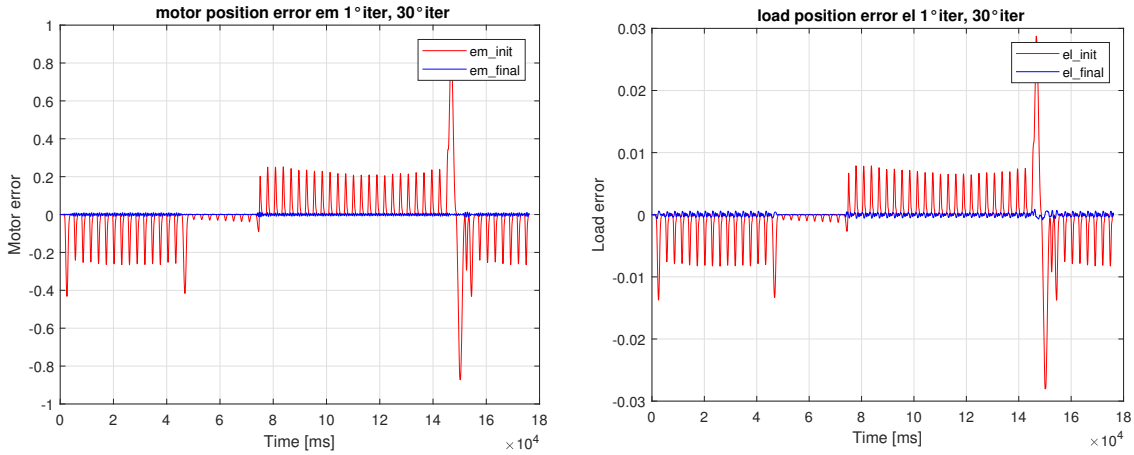

Figure 3.22: Errors and metrics using Data Driven approach on the second trajectory.

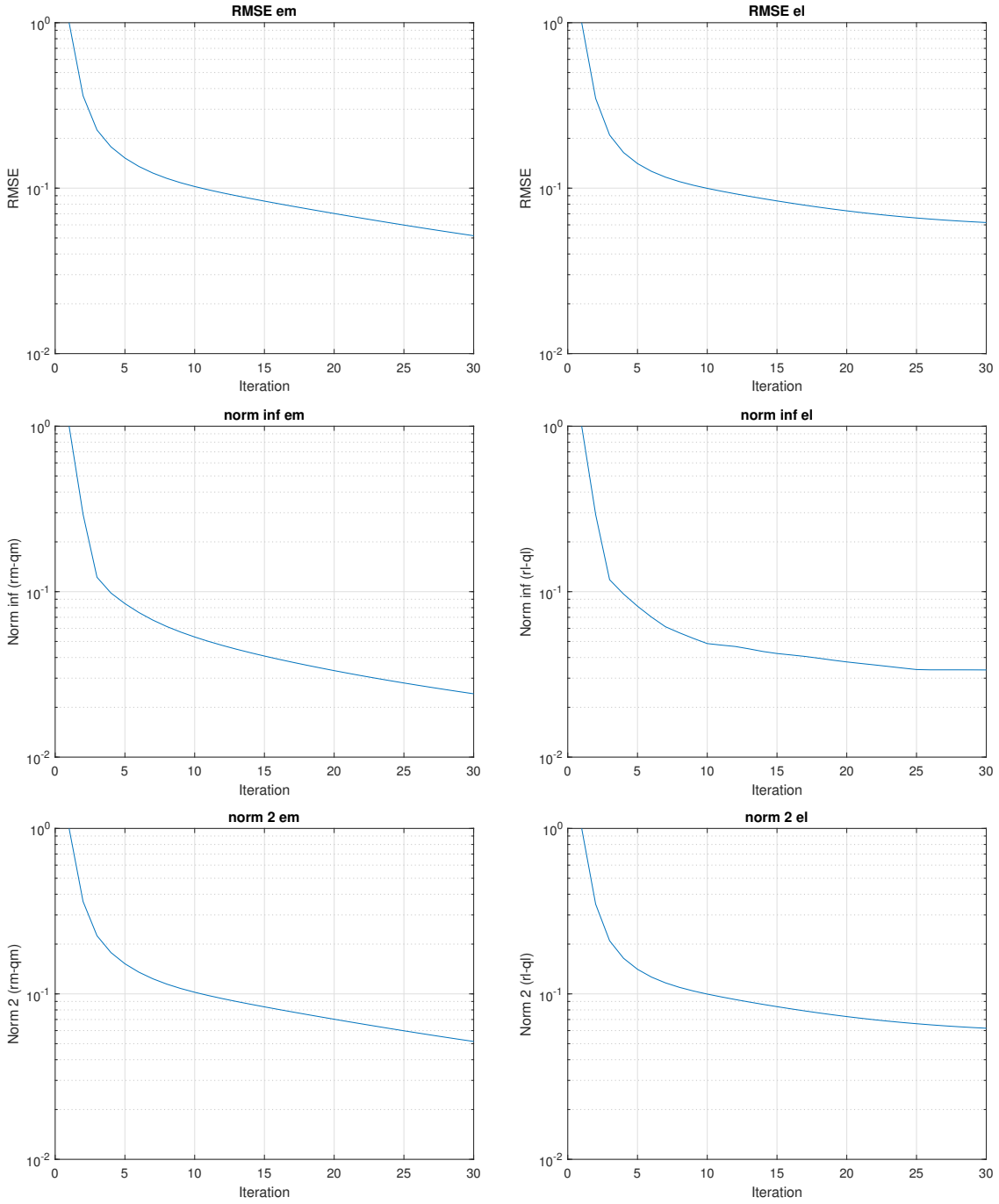

Figure 3.22: Errors and metrics using Data Driven approach on the second trajectory. (cont.)

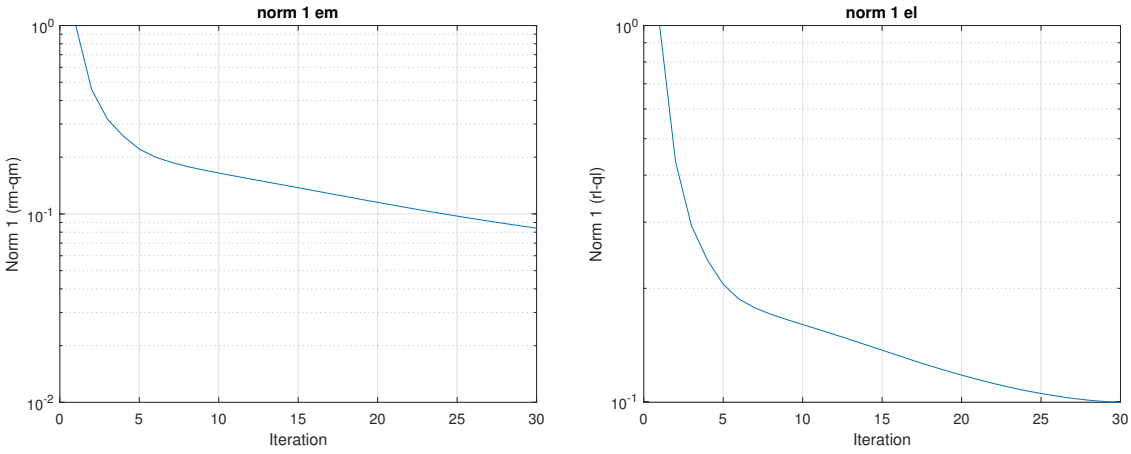

Figure 3.22: Errors and metrics using Data Driven approach on the second trajectory. (cont.)

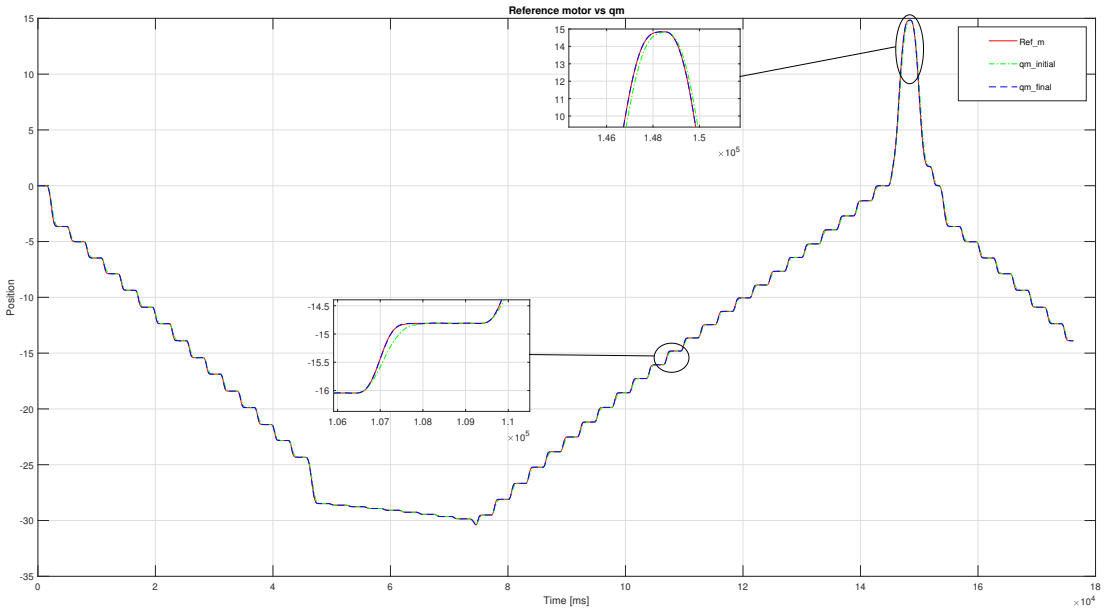

Figure 3.23: Comparison for motor position between the reference (red), the system without ILC (green) and the system plus ILC (blue) with a zoom during an important transient condition (on the bottom) and a less important one (on the top).

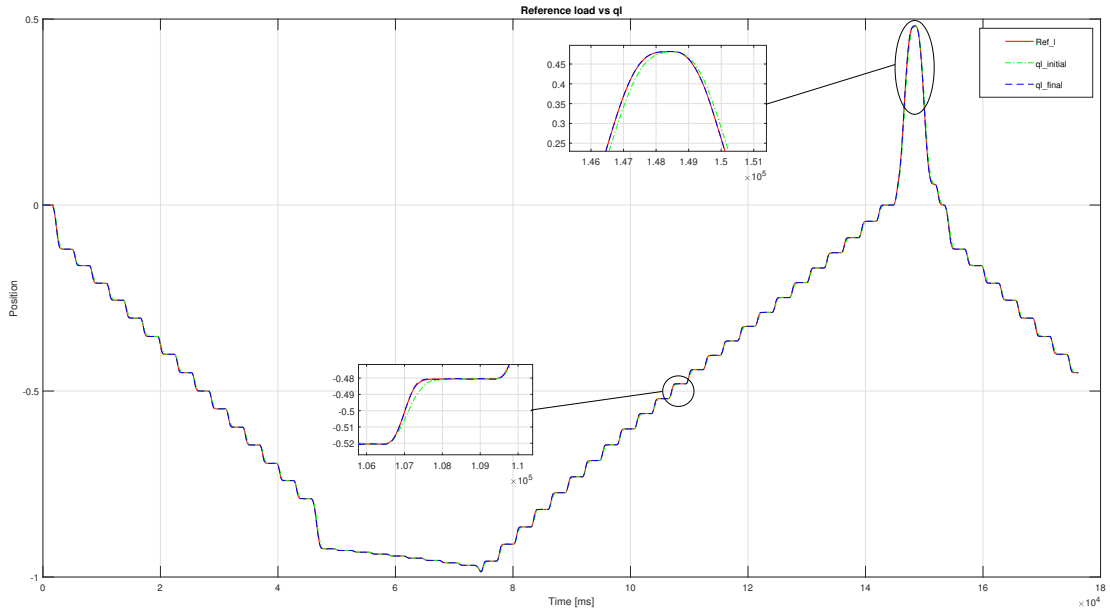

Figure 3.24: Comparison for load position between the reference (red), the system without ILC (green) and the system plus ILC (blue) with a zoom during an important transient condition (on the bottom) and a less important one (on the top).

Proceeding to compute the frequency of the Q-filter, in a similar way to previous cases can be seen that with a value too low of frequency the ILC cannot correctly follow the error and apply a valid correction, instead with a value too high the ILC induces vibrations on the system that lead to instability. To choose the best frequency we consider the global minimization of all the metrics in Figure 3.25 and Figure 3.26. Finally we maintain as cutoff frequency 16Hz, the best one.

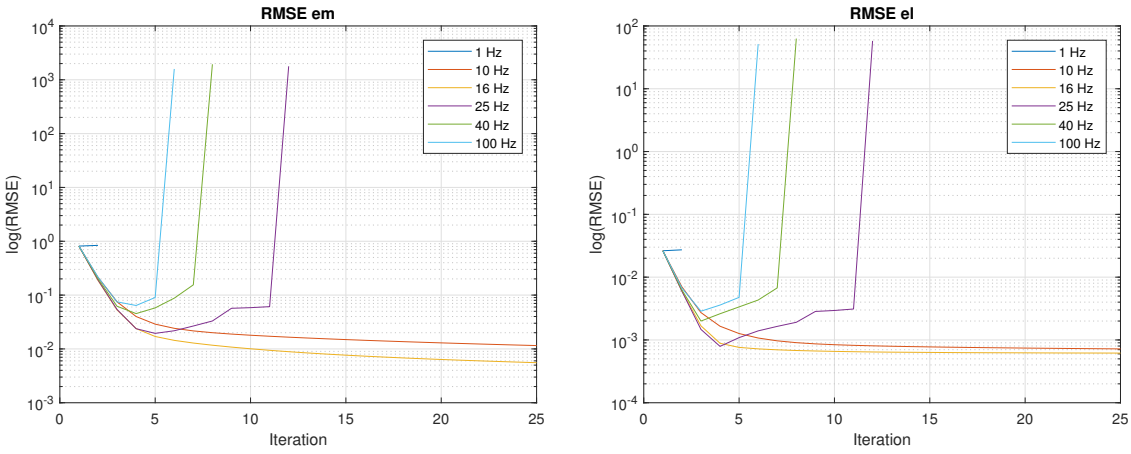

Figure 3.25: For the first trajectory, metrics with different values of the cutoff frequency of the Q-filter.

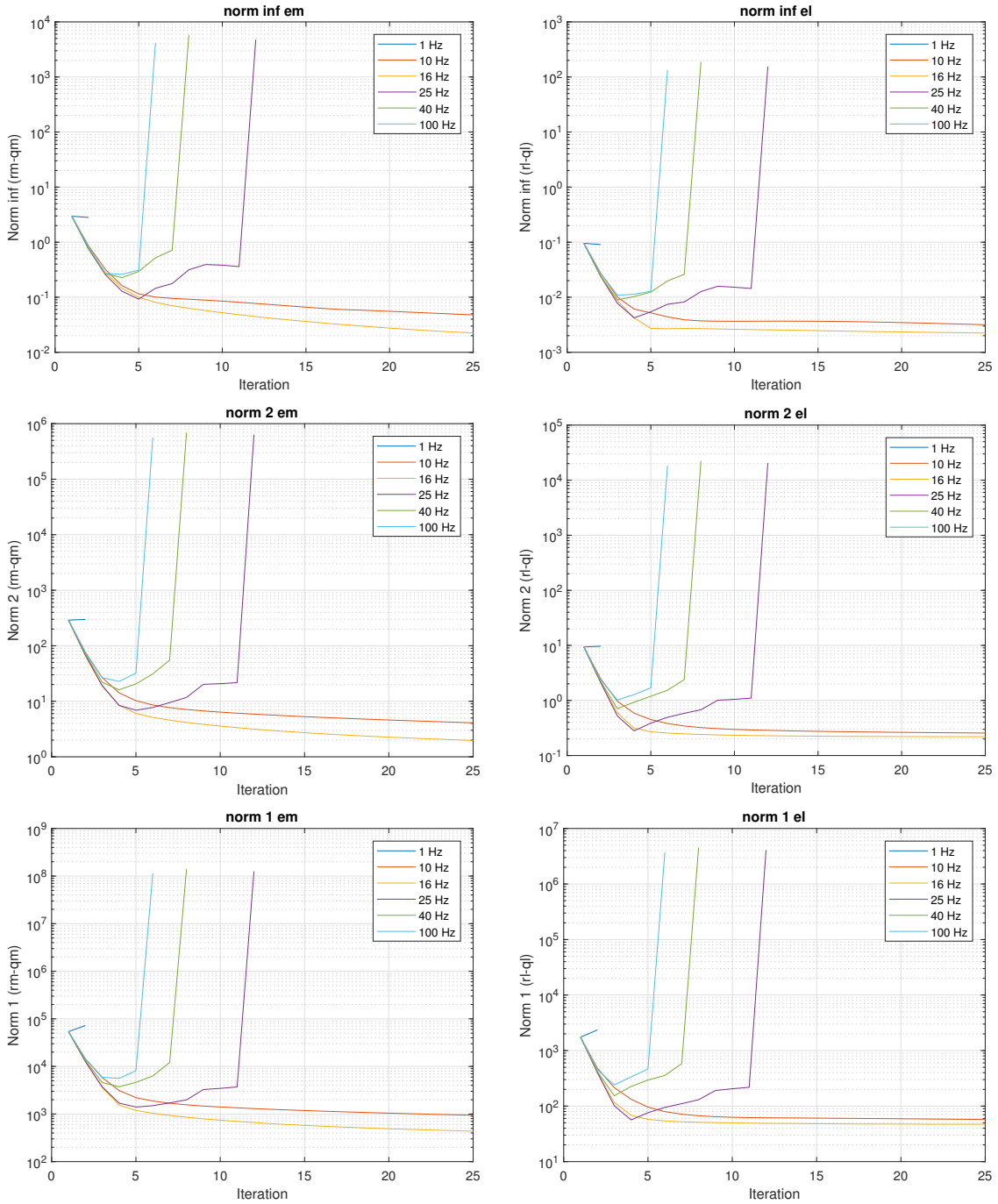

Figure 3.25: For the first trajectory, metrics with different values of the cutoff frequency of the Q-filter. (cont.)

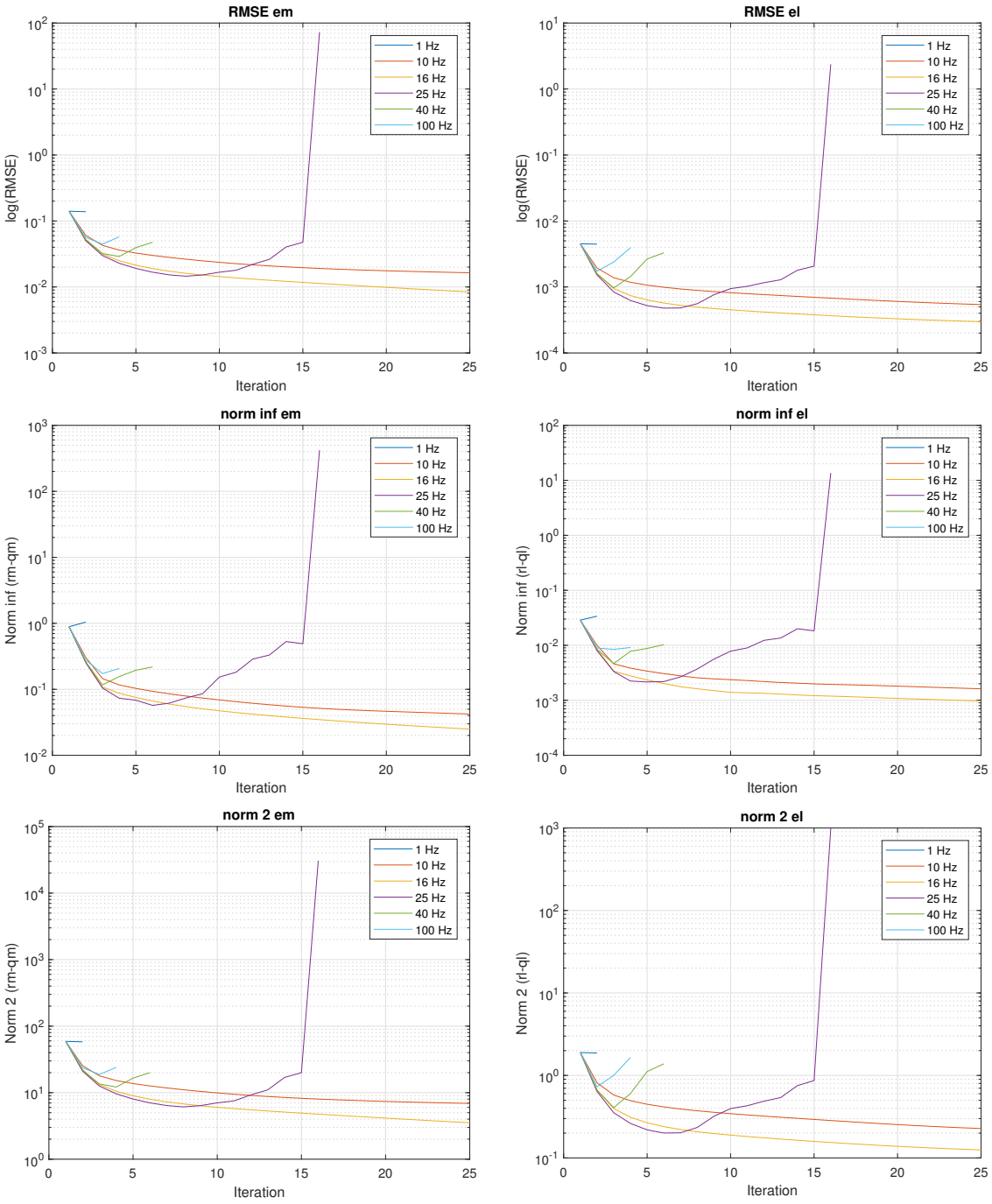

Figure 3.26: For the second trajectory, metrics with different values of the cutoff frequency of the Q-filter.

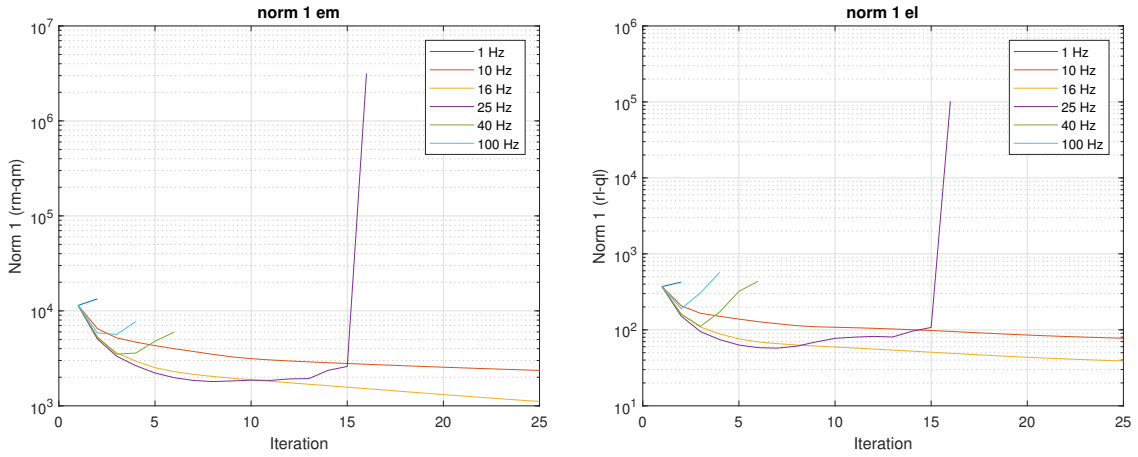

Figure 3.26: For the second trajectory, metrics with different values of the cutoff frequency of the Q-filter. (cont.)

3.4 Conclusion

Comparing the results obtained with the three different approaches in Figure 3.27 and Figure 3.28, it can be seen that the Data Driven ILC is the best solution but takes time to compute the command and requires a precise model of the system like for the plant inversion ILC. Instead the PD-Type, that is the simplest approach, already gives a good solution and it is less time consuming.

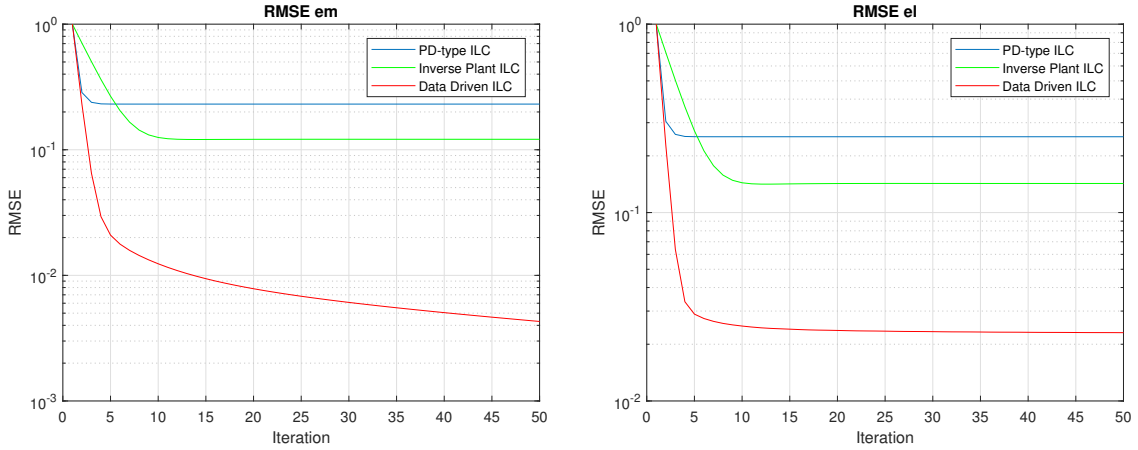

Figure 3.27: First trajectory, comparison between the three approaches.

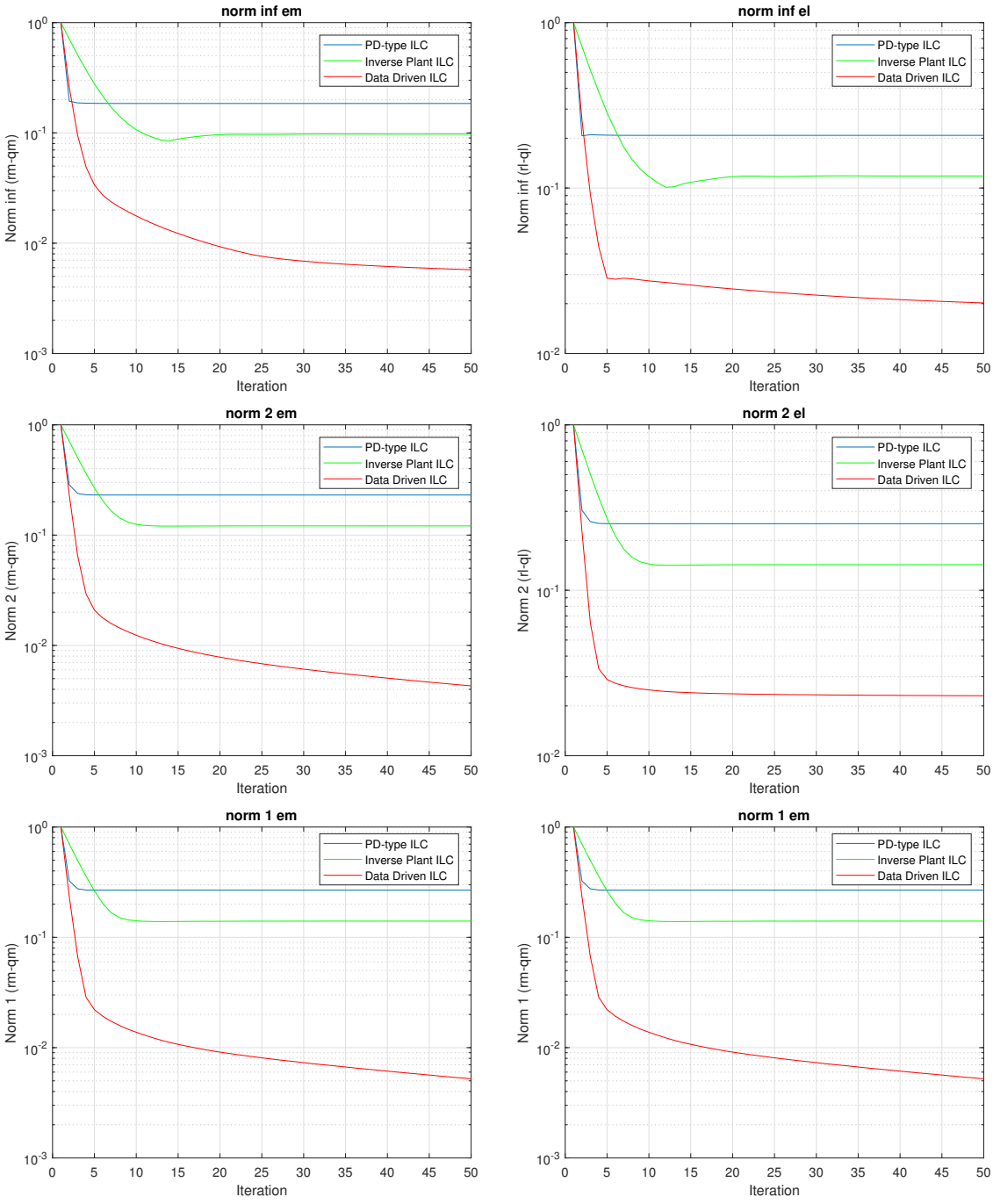

Figure 3.27: First trajectory, comparison between the three approaches. (cont.)

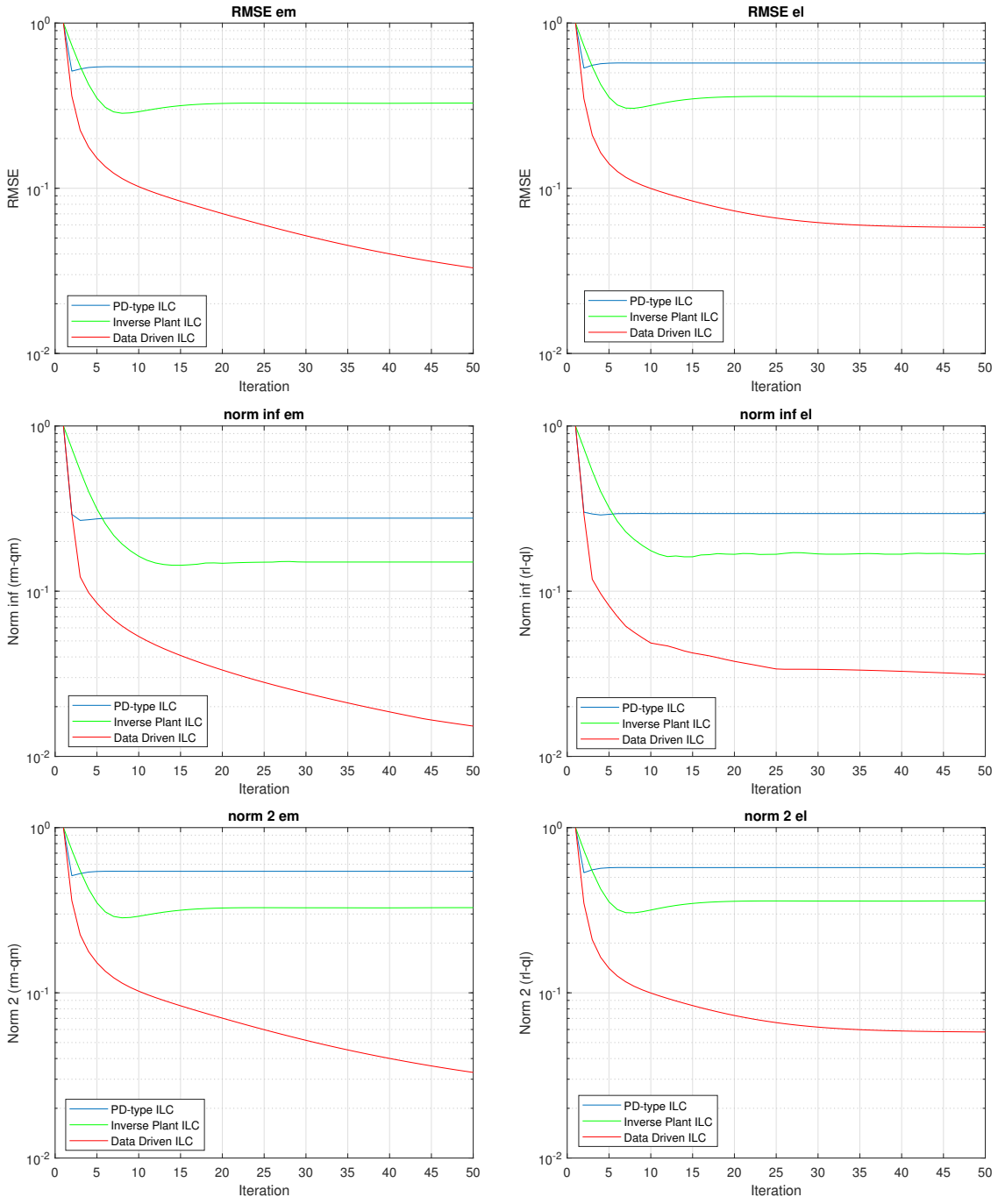

Figure 3.28: Second trajectory, comparison between the three approaches.

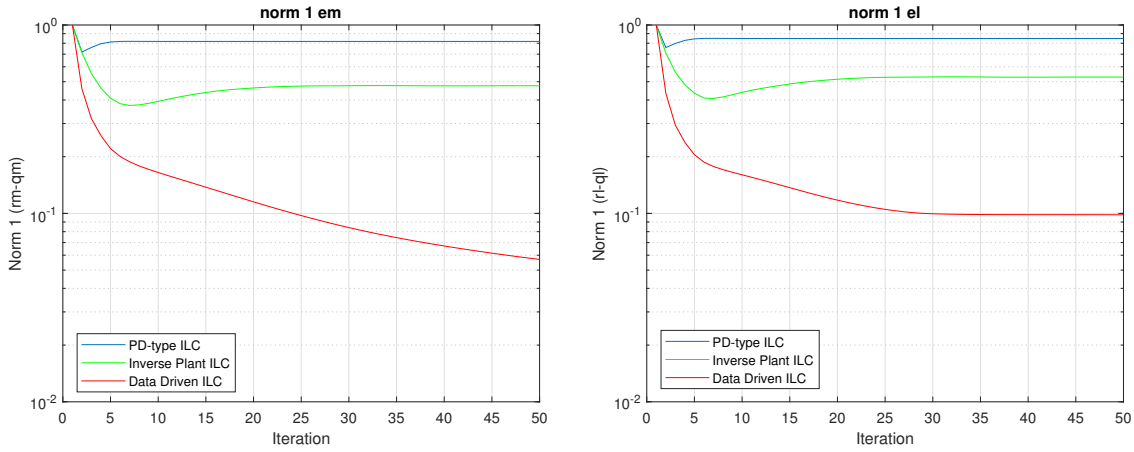

Figure 3.28: Second trajectory, comparison between the three approaches. (cont.)

Chapter 4

Robot Racer 7 - 1.4

In this chapter is shown how the ILC algorithms described in Chapter 2 can be applied to a MIMO system, in particular to the Comau robot *Racer 7 – 1.4*, a 6-DOF industrial manipulator.
The system is simulated through a rigid body model of the robot, reported in the Simulink scheme in Appendix B.5. Without the application of any ILC algorithm, the original control system can be represented as reported in Figure 4.1.

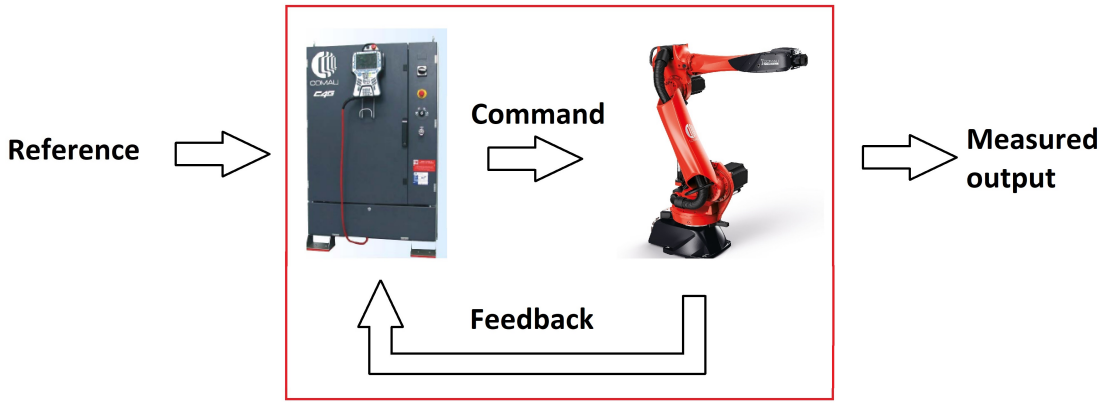

Figure 4.1: Simplified scheme of the original control system.

As shown by the red box, we have in practice only access to the external signals, that are the reference input and the motor output measured by the encoder. We have to work with this black box model because we cannot modify the internal signals of the system, but also because working in this way make easier a possible test of our implementations on the real plant. So, the most appropriate ILC architecture to develop with these working constraints is the serial one (see Figure 4.2).

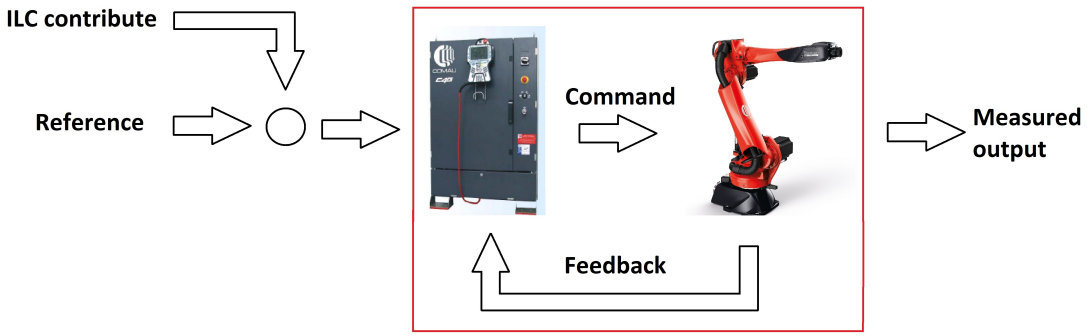

Figure 4.2: Simplified scheme of the control system in ILC serial architecture.

Differently from the previous simulator, only the reference and the error on the motor side are available. The error is computed as difference between the reference on the motor and the measured output of the encoder in the joint space (seeing Appendix B.5, they are reported as *input_q_motor_ref* and *output_q_motor_ref* signals).
Another difference from the previous chapter is the trajectory: in this case it represents the movement of the tool center point during a coating task in the cartesian space (see Figure 4.3). Steady state starting condition are added, prolonging the trajectory, because starting with non-null reference values can make the system unstable.

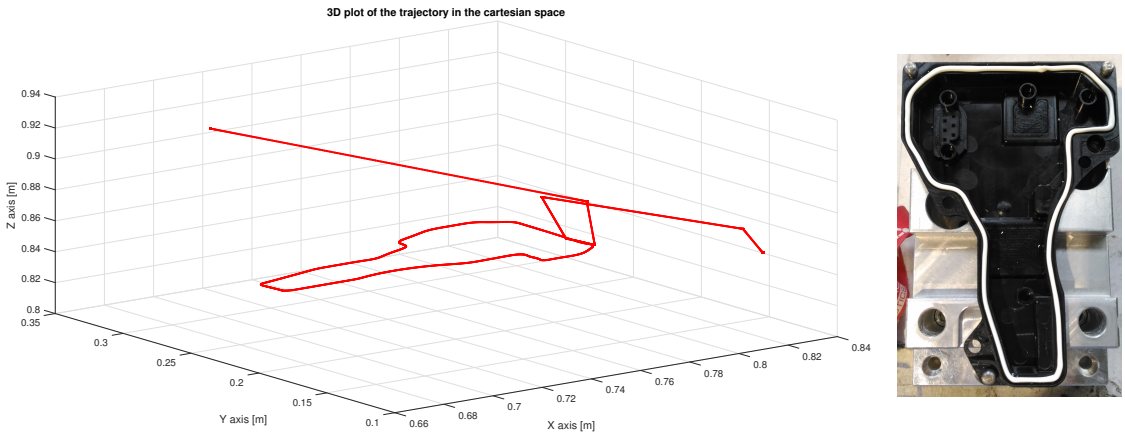

Figure 4.3: On the left, a 3D cartesian plot of the path in the operational space. On the right, a photo of the mechanical piece involved in the coating task.

For the simulation is used the custom load of the task provided by Comau, which mass is equal to 5kg.
In the following subchapters, first is presented the simplest and most model independent ILC algorithm, the PD type, then are presented the results obtained with Data Driven ILC

approach. The MATLAB code is reported respectively in the Appendix A.4 and A.5.
For what concerns the inverse plant ILC, it is not suitable in this situation due to the absence of a model of the entire control system (the red box evidenced in Figure 4.1).

4.1 PD-Type ILC

To develop this kind of algorithm, as already seen in previous chapter, we have to choose the proportional and derivative gain of the error to use in the learning function (2.1) and the cutoff frequency of the Q filter that optimizes the learning transient and steady state behaviour.
In a SISO system, when an initial sub-optimal cutoff frequency of the Q filter is chosen, it is possible to proceed tuning the gains (looking for which minimizes the motor or load errors) and then adjusting again the cutoff frequency of the Q filter to obtain the optimal behaviour with the selected parameters.
In our case, we initially adopt this procedure, by starting to optimize the learning filter for the joint that presents the biggest error. When we proceed to the other joints, we noticed that the optimal parameters for a specific joint hinge the performance of the others, making these parameters not globally the optimal ones. So, in a MIMO system this kind of procedure is not recommended, due to its inefficacy and time-consuming nature.
A global tuning of the parameters that we define, can be obtained adopting two gains for all the joints in the following way:

- Choose a reasonable cutoff frequency, that gives a good result in terms of steady state behaviour for some guessed gains.
- Explore the space of k_p and k_d parameters, by simulating the system response and computing the Euclidean distance between the reference input and measured output, both reported in the operational space. These distances, that should be ideally point by point equal to zero, gives us a metrics of how globally a ILC with certain parameters applied to the six joints is better or worse than another that uses other values.
- Once selected the gains that minimizes the error of the tool center point in the operational space, a tuning of cutoff frequency of the Q filter is done trying to improve the performance.

In our case, we start with Q frequency equal to 200Hz.
We initially explore a wide range of the gains space, then we focalized in a sub-optimal range, that can be visualized in Figure 4.4. The sub-optimality is evidenced from the non-convex nature of this problem, as already find in section 3.1.

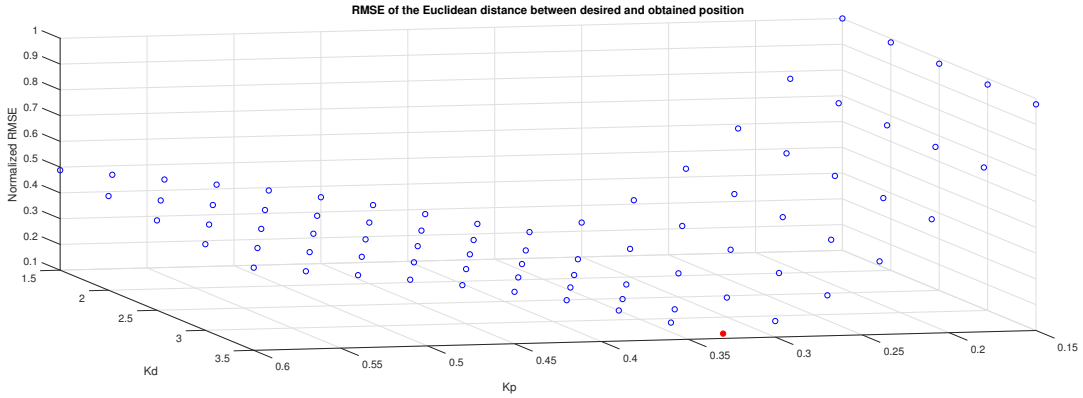

Figure 4.4: Normalized RMSE of the Euclidean distance between the desired and obtained position of the tool center point coordinates, reported as function of the k_p and k_d gains.

All the metrics are calculated in the cartesian space using the Euclidean distance of the tool center point in the desired position from the real position using the ILC. We reported only the RMSE (in Figure 4.4) but all the norms (1, 2, Inf) are minimized in the point of coordinates k_p = 0.33 and k_d = 3.5.
Adopting these last values as sub-optimal parameters, the improvements obtained in terms of our performance indicator can be seen in Figure 4.5, simulating the system for 50 iterations.

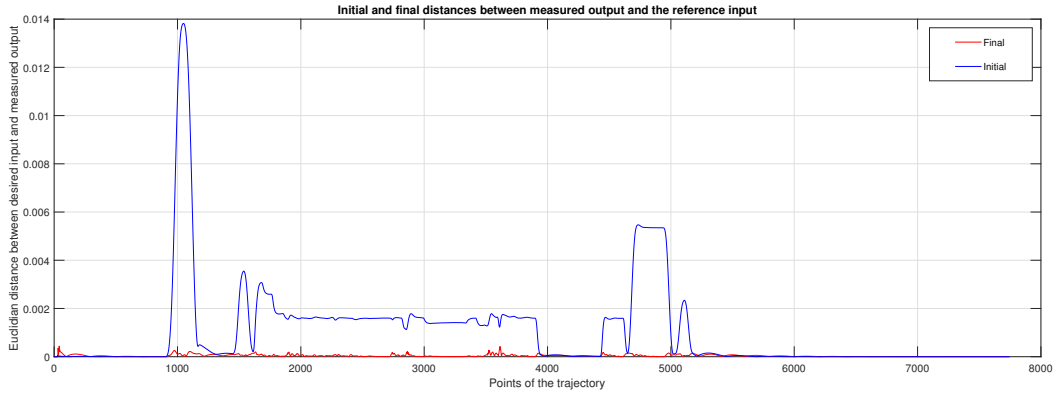

Figure 4.5: Comparison between initial and final distances of measured output from the reference input, computed point by point, in the first and last iteration.

The results of a simulation with the tuned parameters selected are reported in Figure 4.6, Figure 4.7 and Figure 4.8, respectively in terms of errors in the joint space, errors in the operational space and a 3D plot of the obtained cartesian trajectory.

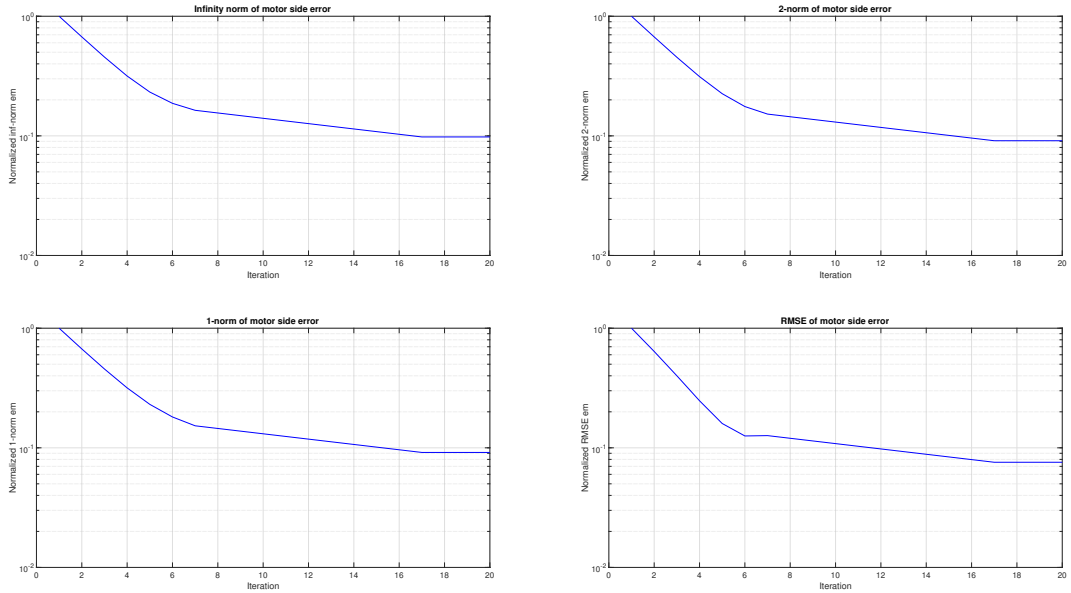

Figure 4.6: Motor side error in the joint space, reported with different type of metrics. Only values for the first 20 iterations (over 50 done) are reported for a better visualization.

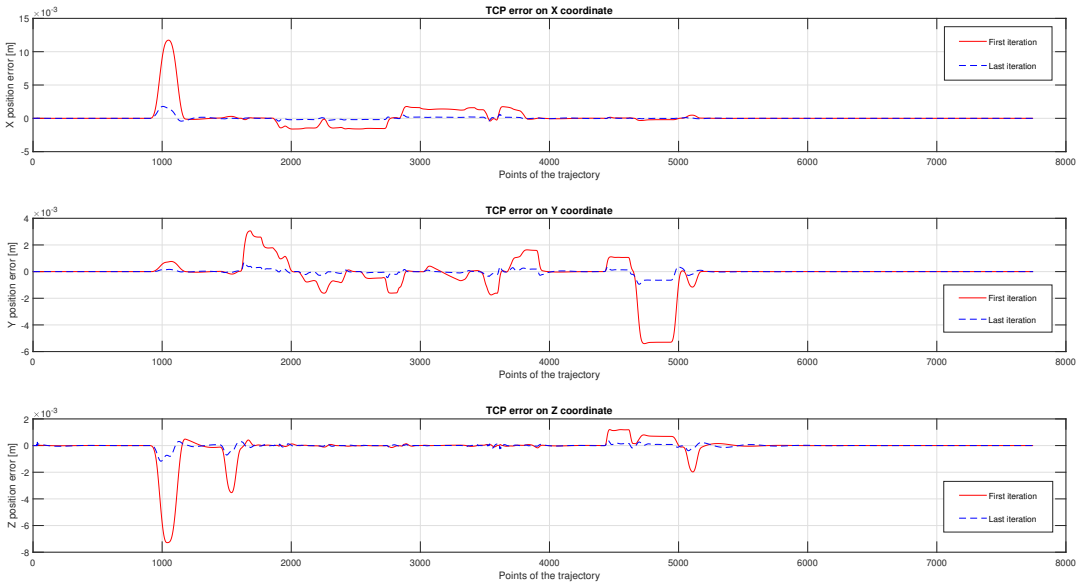

Figure 4.7: Tool center point error, reported in the three different cartesian axis, in the first and last iteration.

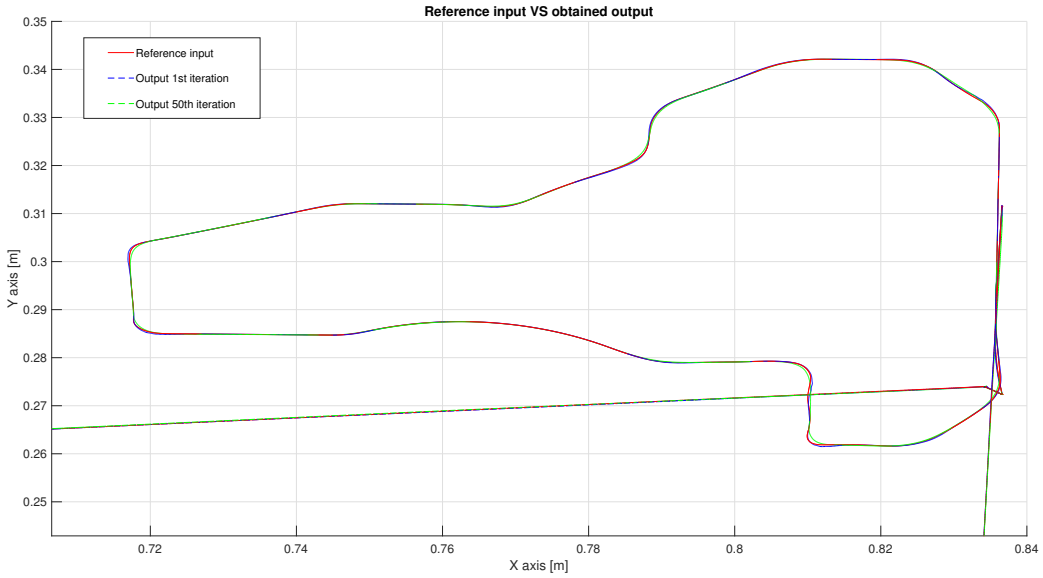

Figure 4.8: Reference input and measured output at the first and last iteration reported in the X-Y plane.

A final refining of the cutoff frequency of the Q filter led us to select a frequency of 200Hz, already adopted in the simulation whose results are reported in Figure 4.6, Figure 4.7 and Figure 4.8. As can be seen in Figure 4.9 in a particular part of the entire path, the Q filter affect the performance in transient condition.

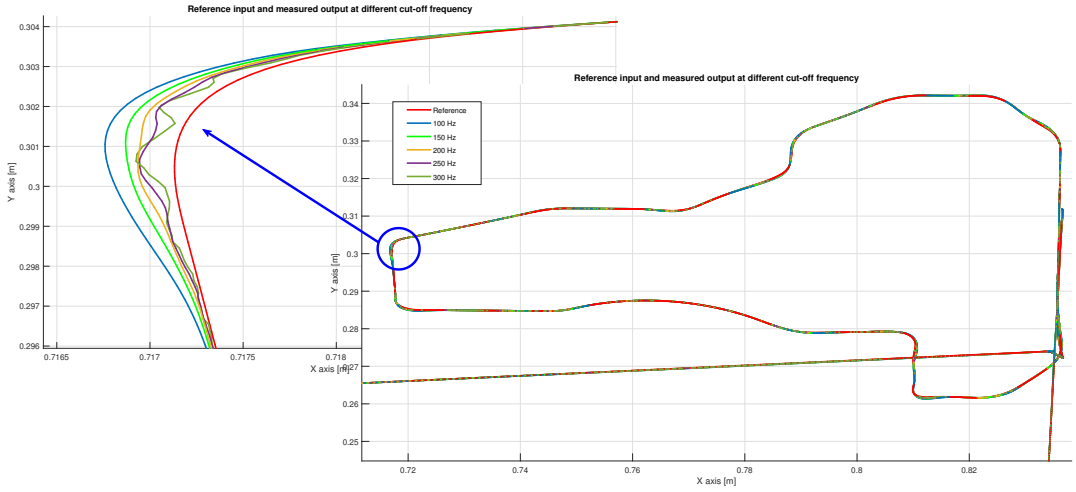

Figure 4.9: Reference input and measured output at different cutoff frequency of the filter Q, reported in the X-Y plane. On the left, a part of the path with important transitory.

Increasing the cutoff frequency of the Q filter makes the ILC faster in tracking signal in important transient condition but oscillatory behaviour arises, so the chosen frequency of 200Hz is a good comprise.

4.2 Data Driven ILC

The first step of this technique is to compute the Markov parameters h, using the theoretical definition reported in the Chapter 3.3 and the mathematical model of the system. In this case, the reduced state space obtained from the linear analysis of the Simulink scheme is a roughly approximation of the system presented, due to its complexity.
The weight factors W_e and W_f (see (2.10)) used are respectively equal to $\mathbf{I}$ and $\mathbf{0}$, as in Chapter 3.3.
The maximum learning gain $\bar{\epsilon}$, defined in (2.10), requires the computation of transpose and 2-norm of large matrix $\boldsymbol{J}$. The total points of the trajectory are 7746 (sampling time equal to $2ms$) and the Markov parameters are a six-tuple for each of the six joints for every point of the trajectory, so the $\boldsymbol{J}$ matrix (defined in (2.4)) is a square matrix of dimension about $47k$ rows and $47k$ columns. Since working with these matrix dimensions can be difficult or very time consuming, we reduce the sequence of Markov parameters: noting that after the first 1000 their values decrease of several orders of magnitude, we try to compute the maximum learning gain increasing the number of the first N value considered.

N	$\left\|\boldsymbol{J}^T\boldsymbol{J}\right\|$	$\bar{\epsilon}$
484	3.574	0.5596
1294	6.085	0.3287
2064	6.628	0.3017
2844	6.810	0.2937

As can be seen in the table reported before, increasing from 2064 to 2844 over 7746 values, the 2-norm does not increase so much because the added contributes are very small respect to the initial ones. So, introducing this approximation we adopt as learning gain

$$\epsilon = 0.5\bar{\epsilon}$$

in order to introducing a safety factor and to make sure anyway asymptotic stability.
The results of a simulation using the parameters described before and the cutoff frequency of the Q filter initially set to 200Hz, are reported in Figure 4.10, Figure 4.11 and Figure 4.12, respectively in terms of errors in the joint space, errors in the operational space and a 3D plot of the obtained cartesian trajectory. Also in this case, we simulate the system for 50 iterations.

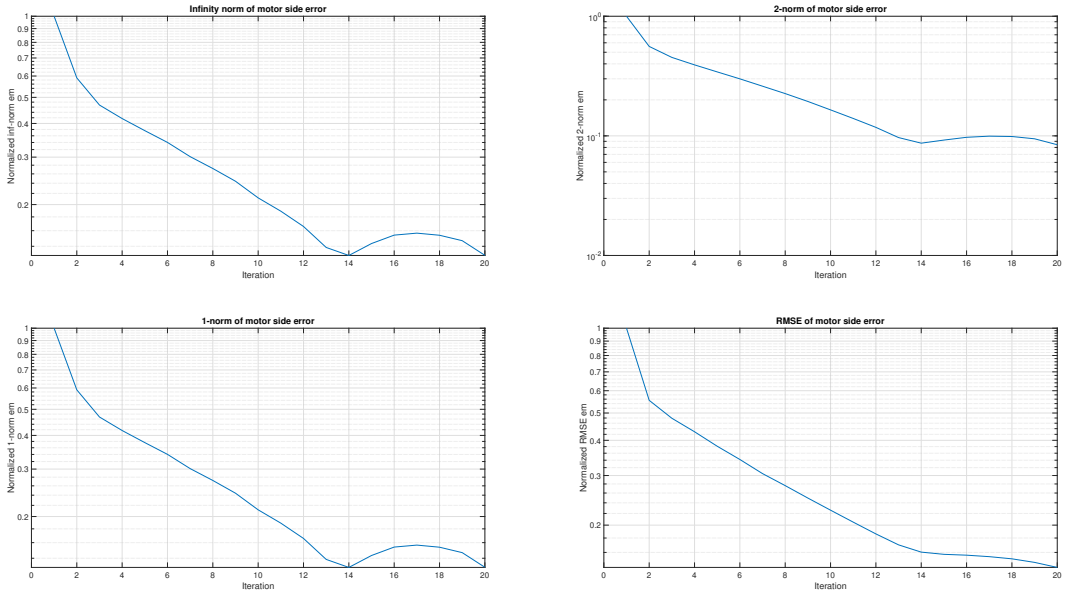

Figure 4.10: Motor side error in the joint space, reported with different type of metrics. Only values for the first 20 iterations (over 50 done) are reported for a better visualization.

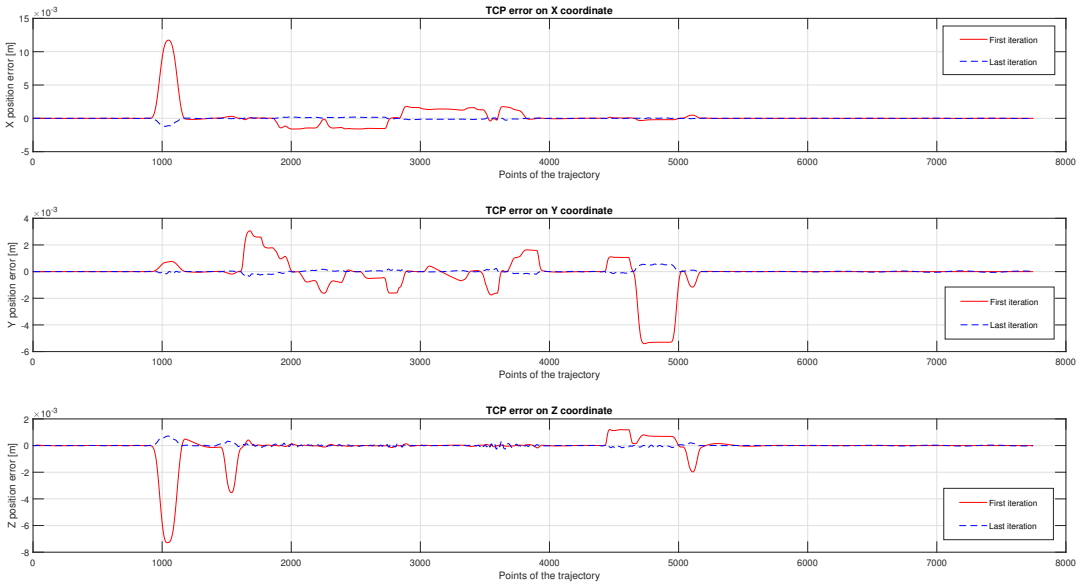

Figure 4.11: Tool center point error, reported in the three different cartesian axis, in the first and last iteration.

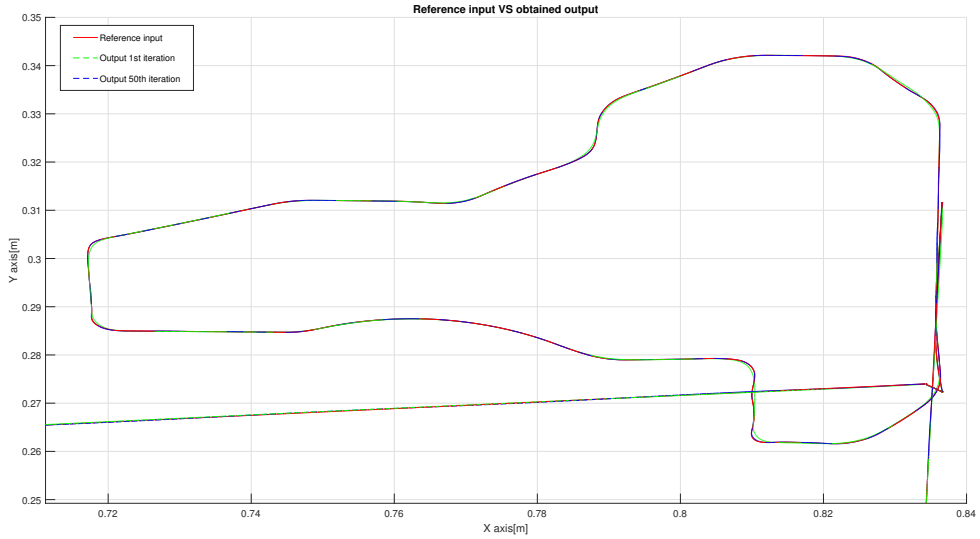

Figure 4.12: Reference input and measured output at the first and last iteration reported in the X-Y plane.

As in section 4.1, the metric adopted to see how the performance of the control system changes using ILC or not is the Euclidean distance between the motor reference position and the simulated one in the cartesian space. It can be seen in Figure 4.13.

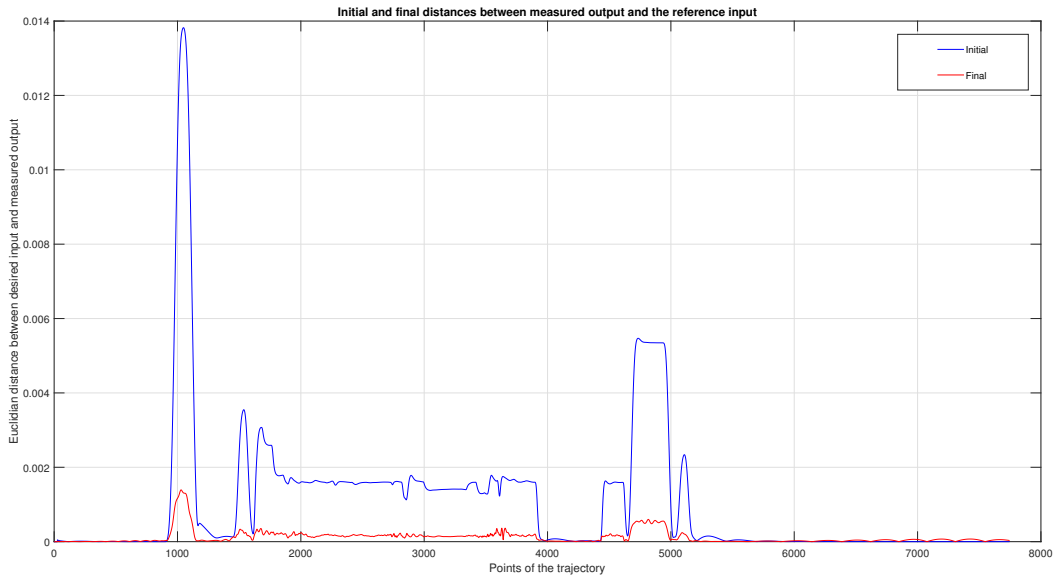

Figure 4.13: Comparison between initial and final distances of measured output from the reference input, computed point by point, in the first and last iteration.

A final tuning of the cutoff frequency of the Q filter led us to confirm a frequency of 200 Hz as the best one, already adopted in the simulation whose results are reported in Figure 4.10, Figure 4.11 and Figure 4.12. As can be seen in Figure 4.14 in a particular portion of the entire path, the Q filter hinge the performance of the ILC corrections in transient condition.

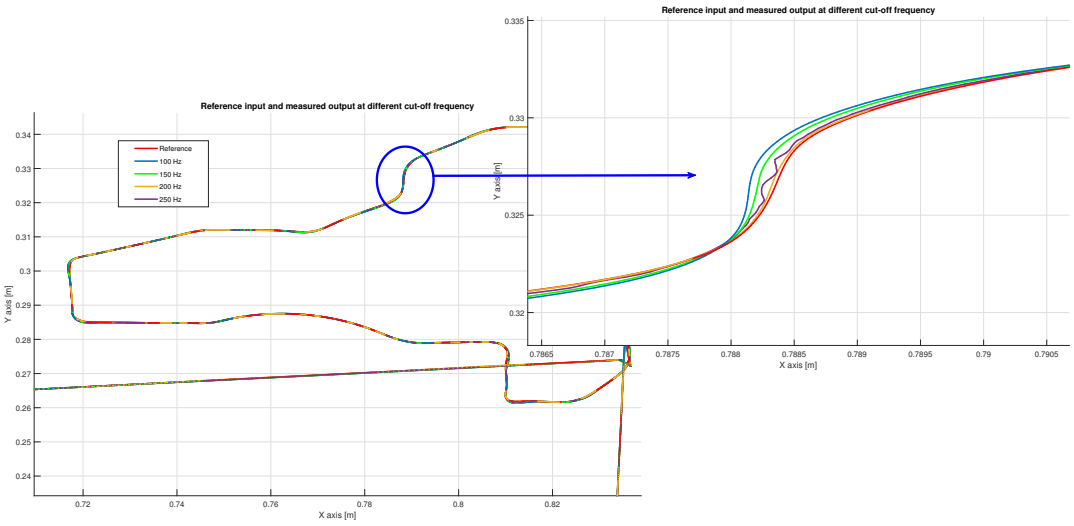

Figure 4.14: Reference input and measured output at different cutoff frequency of the filter Q, reported in the X-Y plane. On the right, a part of the path with important transitory.

Increasing the cutoff frequency of the Q filter, as in the previous case, oscillatory behaviour arises, so the chosen frequency of 200Hz is a good comprise.

4.3 Conclusion

Between the two kinds of ILC developed, the best results are obtained with PD-Type ILC. An evidence of this can be seen comparing the results obtained in terms of Euclidean distance between reference input and obtained output position in the operative space of the tool center point (see Figure 4.15).

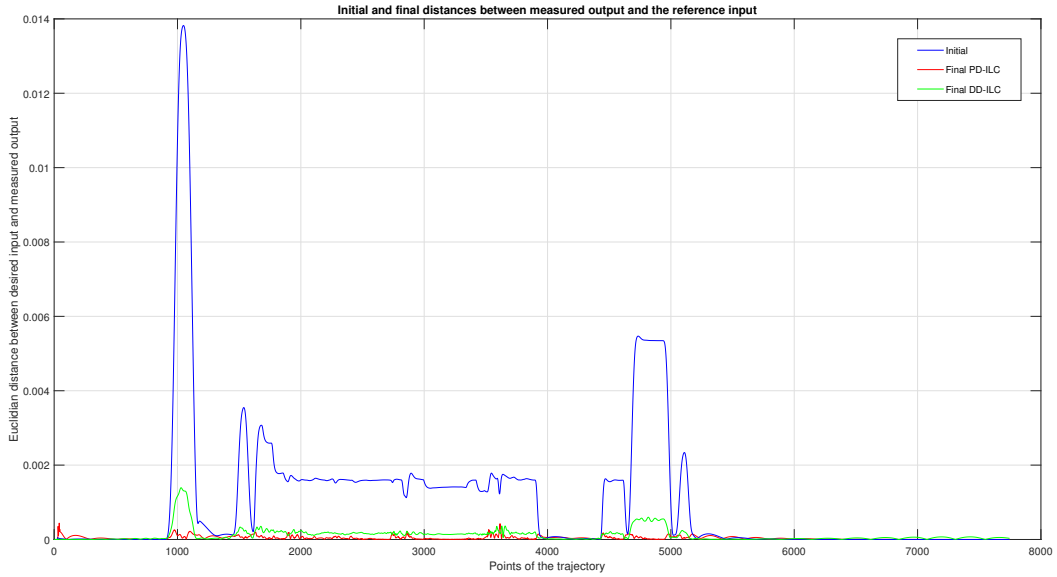

Figure 4.15: Comparison between distances from the desired position at the first iteration and at the last (50^{th}) iteration using PD-Type ILC and DD-ILC

Another way to compare the two methods is to see the error metrics at steady state condition. Also in this case, as can be seen from Figure 4.16, the best results are obtained with the PD-Type ILC except for the 2-norm where the error is comparable with the DD-ILC.

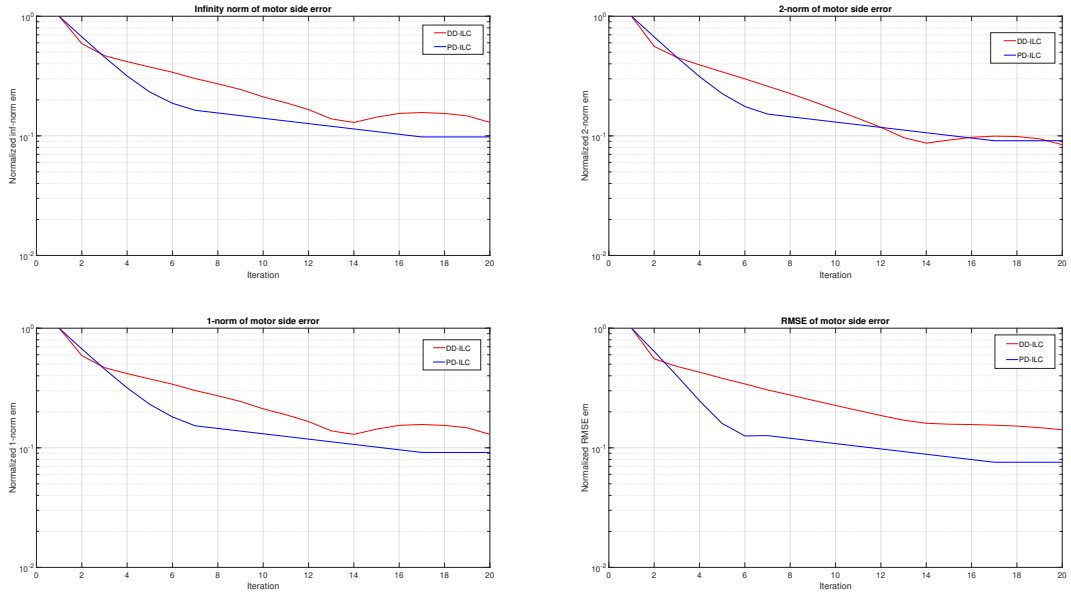

Figure 4.16: Comparison between motor side error in the joint space, reported with different type of metrics, obtained with PD-ILC and DD-ILC.

Differently from what is obtained with the first simulator, the Data Driven ILC is not performant as in that case. We expect these results, as already said, due to the Markov parameters computation: they are not found directly identifying the $\boldsymbol{J}$ matrix as in [16], but we compute them through theoretical definition using a linearised reduced model. Considering that the control system, differently from the simulator used in Chapter 3, is more complex, the associated reduced state space is too approximated.

Chapter 5

Conclusion

In this thesis we analysed the basis of the Iterative Learning Control and, for two Comau manipulators, three possible implementations: PD-ILC, Inverse Plant ILC and Data Driven ILC. Problems in stability, performance, learning transient behaviour and robustness were discussed along with the design techniques. The ability of ILC to use available measurements in an offline learning manner, coupled with feedback-feedforward controller robustness, enables an effective tracking control solution. The effectiveness depends on many factors: the tuning of the parameters for PD-ILC, the Q-filter frequency, the accuracy of the model used for the plant inversion or the estimation of the Markov parameters for the Data Driven are some examples. Each system has to be analysed case-by-case because, as seen in the previous chapters, a solution or an implementation can be optimal for a system but not for another one.
During the thesis some questions arise and in future works can be deepened, these include:

1. Independence from the trajectory of the parameters in the PD-ILC: in first approximation we consider the parameters trajectory independent. In the robot *NJ4 220 – 2.4* we see that the optimal parameter for the first trajectory are good also for the second one, so we can suppose that they depend minimally from the path. It is possible to demonstrate it by the tuning of the parameters also for other trajectories and comparing the possible improvements respect adopting the same parameters.

2. Difference between results for robot *NJ4 220 - 2.4* and robot *Racer 7 – 1.4*: as previously said each system has to be analysed case-by-case. In the first simulator the DD-ILC is better than PD-ILC and Inverse Plant ILC instead in the second one the PD-ILC is better than DD-ILC and even Inverse Plant ILC cannot be evaluated. Because the second robot simulator is more complex the linearized model is worse than in the first simple case and so the effectiveness of the Data Driven is reduced.

3. Importance of a good model identification for Inverse Plant ILC and DD-ILC: as can be seen by the results of the two simulators, the model plays an important role for the performance of the ILC, in particular for the Inverse Plant and Data Driven ILC.

Without a good model the inverse plant or the Markov parameters lead to a wrong reference correction and an ineffective ILC. To solve this problem in other papers they identify the model of the system.

4. Independence between joint parameters for PD-ILC in MIMO systems: in the second simulator we start to optimize the parameters for each single joint but when we proceed to other joints we noticed that the optimal values for a specific joint hinge the performance of other making these parameters not the optimal ones. We define a global tuning of the parameters adopting two gains for all the joints.

5. Difference between applying ILC correction in the operative space or in the joint space: all the results reported are obtained applying the ILC correction in the joint space. To be able to apply the correction in the operative space we have first to obtain the error or the variable of interest in the cartesian space using the direct kinematics and then compute the ILC corrections and report it in the joint space using the inverse kinematics. This last operation is very time consuming. Alternatively, we can use the Jacobian matrix to make the conversion between the joint and operative space faster, approximating a small difference in position in discrete time as a velocity. Unfortunately, the error that we introduced by this approximation is similar to the position error that we want to correct, so the results obtained are shoddy.

6. Importance of performance metrics reduction in joint space and operative space: during the simulation of the various ILC types we observe that only in some particular conditions the performance metrics in the operative space improve while in the joint space remain stable or increase and viceversa. In principle, if we obtain better results in one of the space we have almost always improvements in the other one. Based on which space we are interested we have to focus in particular on the performance in that space; in almost of all the cases is the cartesian space because is where the task is originally defined. Considering these aspects, worse performance in the joint space are not to be considered if accompanied with improvements in the operative space.

7. Use of real exteroceptive sensors instead of simulated proprioceptive ones: during the thesis, for robot *Racer 7 – 1.4*, we concentrate to improve the trajectory on motor side calculating the error using the position from the simulated motor sensor. To improve the result of the ILC on the link side we have to use external sensors like accelerometer or camera to acquire the real position of the tool center point of robot in the cartesian space. In a similar way we can apply the ILC to the link position.

In short, ILC is an interesting approach to control based on the idea that the input reference can be refined trial by trial using the errors recorded in the past iterations to obtain the desired output. It has numerous areas of application such as robotics or in general driver controller and so it has a promising future for continued research and improvements.

Appendix A

Matlab Code

Listing A.1: ILC Simulation code Robot NJ4 220 - 2.4

```
% Simulation Code

clear all
close all
clc

%% Data robot initialization

data_ax4_nj_4220

%% Trajectory generation

genera_traj;

%% Simulink Scheme

robot_equation

%% Definition/computation L filter

inverse_ = 0;
ddilc_ = 1;

if inverse_
    inverse_plant
elseif ddilc_
    load h.mat
    we = 1;
    epsi = 0.8;
else
    Kp_ilc = 1;
```

```
32      Kd_ilc = 5.8;
33      L = zpk(Kp_ilc*z+Kd_ilc*(z-1));
34  end
35  close all
36
37  %% Q Filter
38
39  filter_index = 1;     %1 for Qz, 2 for input filter, 3 no filter
40
41  %Qs and Qz definition
42  ws_ilc = Cut_Off_frq;
43  Qs = tf(1, [1/ws_ilc^2 sqrt(2)/ws_ilc 1]);
44  [num_Qs, den_Qs] = tfdata(Qs,'v');
45  Qz = c2d(Qs,Ts,'zoh');
46  [num_Qz, den_Qz] = tfdata(Qz,'v');
47
48  %% ILC INITIALIZATION
49
50  l = length(trajectory);
51  delta = 1;
52  itermax = 50;
53  n = 1;
54
55  % initialization of each cicle
56  em_z = zeros(l,itermax);
57  el_z = zeros(l,itermax);
58  em_norm_inf = zeros(1,itermax);
59  em_norm_2 = zeros(1,itermax);
60  em_norm_1 = zeros(1,itermax);
61  el_norm_inf = zeros(1,itermax);
62  el_norm_2 = zeros(1,itermax);
63  el_norm_1 = zeros(1,itermax);
64  RMSE_motor = zeros(1,itermax);
65  RMSE_load = zeros(1,itermax);
66  coef_rm_sim = timeseries(zeros(l,1));
67  coef_rm = zeros(l,itermax);
68  z_inv = zeros(1,l);
69  rm_ilc_sim = timeseries(zeros(l,1));
70
71  for iter = 1:itermax
72
73      % display iteration number
74      iter
75      sim('System_scheme.mdl')
76
77      if iter == 1
78          qm_first_iter = qm.signals.values;
79          ql_first_iter = ql.signals.values;
80      end
81
```

```
82      % ILC-reference to update
83      if inverse_
84          coef_rm(:,iter+1) = rm_ilc.signals.values + ...
                lsim(L,em.signals.values);
85      elseif ddilc_
86          ddilc
87      else
88          coef_rm(:,iter+1) = rm_ilc.signals.values
89              + Kp_ilc*[em.signals.values(delta+1:end,1);
90                  zeros(delta,1)]
91              + Kd_ilc*([em.signals.values(delta+1:end,1);
92                  zeros(delta,1)] - em.signals.values);
93      end
94
95      coef_rm_sim = timeseries(coef_rm(:,iter+1),Time);
96
97      % data for 3D plot of em and el
98      iter_x = ones(l,1)*(1:itermax);
99      em_z(:,iter) = em.signals.values;
100     el_z(:,iter) = el.signals.values;
101
102     % norm1,2,Inf error for motor side
103     em_norm_1(iter) = norm(em.signals.values,1)
104     em_norm_2(iter) = norm(em.signals.values,2)
105     em_norm_inf(iter) = norm(em.signals.values,Inf)
106
107     % norm1,2,Inf error for load side
108     el_norm_1(iter) = norm(el.signals.values,1)
109     el_norm_2(iter) = norm(el.signals.values,2)
110     el_norm_inf(iter) = norm(el.signals.values,Inf)
111
112     %RMSE
113     RMSE_motor(iter) = ...
            sqrt(immse(Ref.signals.values,qm.signals.values))
114     RMSE_load(iter) = ...
            sqrt(immse(1/Ntr*2*pi*Ref.signals.values,ql.signals.values))
115
116 end
117
118 %% Normalization
119
120 normalized = 1;
121 if normalized
122     em_norm_inf = em_norm_inf/max(em_norm_inf);
123     em_norm_2 = em_norm_2/max(em_norm_2);
124     em_norm_1 = em_norm_1/max(em_norm_1);
125     el_norm_inf = el_norm_inf/max(el_norm_inf);
126     el_norm_2 = el_norm_2/max(el_norm_2);
127     el_norm_1 = el_norm_1/max(el_norm_1);
128     RMSE_motor = RMSE_motor/max(RMSE_motor);
```

```
129         RMSE_load = RMSE_load/max(RMSE_load);
130 end
131
132 %% plot
133
134 linear = 0;
135 % RMSE motor and load
136 if linear
137         fig1 = figure(1); plot(RMSE_motor(1:iter)), ...
                title('RMSE(rm-qm)'), grid on
138         fig2 = figure(2); plot(RMSE_load(1:iter)), ...
                title('RMSE(rl-ql)'), grid on
139 else
140         fig1 = figure(1); semilogy(RMSE_motor(1:iter)), ...
                title('RMSE(rm-qm)'), grid on
141         fig2 = figure(2); semilogy(RMSE_load(1:iter)), ...
                title('RMSE(rl-ql)'), grid on
142 end
143
144 % error every iteration
145 fig3 = figure(3); plot3(iter_x, 1:1:l, em_z), title('motor ...
        position error rm-qm'), grid on
146 fig4 = figure(4); plot3(iter_x, 1:1:l, el_z), title('load ...
        position error rl-ql'), grid on
147
148 % error first and last iteration
149 fig5 = figure(5); plot(em_z(:,1),'r'), hold on, ...
        plot(em_z(:,iter),'b'), title(sprintf('motor position error ...
        rm-qm 1 iter, %d iter', iter)), grid on, ...
        legend('em\_init','em\_final')
150 fig6 = figure(6); plot(el_z(:,1),'r'), hold on, ...
        plot(el_z(:,iter),'b'), title(sprintf('load position error ...
        rl-ql 1 iter, %d iter', iter)), grid on, ...
        legend('el\_init','el\_final')
151 fig7 = figure(7); plot(abs(em_z(:,1)),'r'), hold on, ...
        plot(abs(em_z(:,iter)),'b'), title(sprintf('abs motor ...
        position error |rm-qm| 1 iter, %d iter', iter)), grid on, ...
        legend('em\_init','em\_final')
152 fig8 = figure(8); plot(abs(el_z(:,1)),'r'), hold on, ...
        plot(abs(el_z(:,iter)),'b'), title(sprintf('abs load position ...
        error |rl-ql| 1 iter, %d iter', iter)), grid on, ...
        legend('el\_init','el\_final')
153
154 % initial reference vs final qm or ql
155 fig9 = figure(9); plot(Ref.signals.values,'r','LineWidth',1), ...
        hold on, plot(qm_first_iter,'-.g','LineWidth',1), ...
        plot(qm.signals.values,'--b','LineWidth',1), title('Reference ...
        motor vs qm'), grid on, ...
        legend('Ref\_m','qm\_initial','qm\_final')
```

```
156 fig10 = figure(10); ...
        plot(Ref.signals.values/Ntr*2*pi,'r','LineWidth',1), hold on, ...
        plot(ql_first_iter,'-.g','LineWidth',1), ...
        plot(ql.signals.values,'--b','LineWidth',1), title('Reference ...
        load vs ql'), grid on, ...
        legend('Ref\_l','ql\_initial','ql\_final')
157
158 % error norm inf and 2
159 if linear
160     fig11 = figure(11); plot(em_norm_inf(1:iter)), ...
            title('infinity norm em=rm-qm'), grid on
161     fig12 = figure(12); plot(em_norm_2(1:iter)), title('2-norm ...
            em=rm-qm'), grid on
162     fig13 = figure(13); plot(em_norm_1(1:iter)), title('1-norm ...
            em=rm-qm'), grid on
163     fig14 = figure(14); plot(el_norm_inf(1:iter)), ...
            title('infinity norm el=rl-ql'), grid on
164     fig15 = figure(15); plot(el_norm_2(1:iter)), title('2-norm ...
            el=rl-ql'), grid on
165     fig16 = figure(16); plot(el_norm_1(1:iter)), title('1-norm ...
            el=rl-ql'), grid on
166 else
167     fig11 = figure(11); semilogy(em_norm_inf(1:iter)), ...
            title('infinity norm em=rm-qm'), grid on
168     fig12 = figure(12); semilogy(em_norm_2(1:iter)), ...
            title('2-norm em=rm-qm'), grid on
169     fig13 = figure(13); semilogy(em_norm_1(1:iter)), ...
            title('1-norm em=rm-qm'), grid on
170     fig14 = figure(14); semilogy(el_norm_inf(1:iter)), ...
            title('infinity norm el=rl-ql'), grid on
171     fig15 = figure(15); semilogy(el_norm_2(1:iter)), ...
            title('2-norm el=rl-ql'), grid on
172     fig16 = figure(16); semilogy(el_norm_1(1:iter)), ...
            title('1-norm el=rl-ql'), grid on
173 end
```

Listing A.2: Inverse Plant code Robot NJ4 220 - 2.4

```
1  % Inverse plant
2
3  G = MotPosition
4  [numG, denG] = tfdata(G,'v');
5  G_z1 = (tf(numG, denG, Ts,'Variable','z^-1'));
6  [numGz1, denGz1] = tfdata(G_z1,'v');
7  delay = length(numGz1(numGz1==0));
8  numBcz1 = numGz1(numGz1 =0);
9  Bc_z1 = (tf(numBcz1,1,Ts,'Variable','z^-1'));
10 Bc_z = (tf(numG,1,Ts));
11 Ac_z1 = (tf(denGz1,1,Ts,'Variable','z^-1'));
```

```
12 Ac_z = (tf(denG,1,Ts));
13
14 G_z1_b = z^-delay*Bc_z1/Ac_z1
15 [numGz1b, denGz1b] = tfdata(G_z1_b,'v');
16
17 zeri = zero(Bc_z1);
18 Bca_z1 = tf(1,1,Ts,'Variable','z^-1');
19 Bcu_z1 = tf(1,1,Ts,'Variable','z^-1');
20
21 i = 1;
22 % Stable*Unstable decomposition
23 while i <= length(zeri)
24     if abs(zeri(i)) < 1
25         if(imag(zeri(i)) == 0)
26             Bca_z1 = Bca_z1 * ...
                   tf(poly([zeri(i)]),1,Ts,'Variable','z^-1');
27             i = i+1;
28         else
29             Bca_z1 = Bca_z1 * ...
                   tf(poly([zeri(i);zeri(i+1)]),1,Ts,'Variable','z^-1');
30             i = i+2;
31         end
32     else
33         if(imag(zeri(i)) == 0)
34             Bcu_z1 = Bcu_z1 * ...
                   tf(poly([zeri(i)]),1,Ts,'Variable','z^-1');
35             i = i+1;
36         else
37             Bcu_z1 = Bcu_z1 * ...
                   tf(poly([zeri(i);zeri(i+1)]),1,Ts,'Variable','z^-1');
38             i = i+2;
39         end
40     end
41 end
42
43
44 [ign,ign,Bcu_K] = zpkdata(Bc_z1);
45 Bc_z1_b = tf(Bca_z1 * Bcu_z1 * Bcu_K);
46
47 [numBcaz1, ign] = tfdata(Bca_z1,'v');
48 Bca_z = (tf(numBcaz1,1,Ts));
49
50 [numBcuz1, ign] = tfdata(Bcu_z1,'v');
51 Bcuz1K = ddcgain(numBcuz1,1);
52 Bcu_z = (tf(numBcuz1,1,Ts));
53 Bcu_z1s = (tf(fliplr(numBcuz1),1,Ts,'Variable','z^-1'));
54
55 G_inv = z^-delay*(Ac_z1 * Bcu_z1) / (Bca_z1 * (Bcuz1K)^2);
56
57 L = G_inv
```

Listing A.3: Data Driven code Robot NJ4 220 - 2.4

```
%ddilc

e_ilc = em.signals.values;
e_ilc_inv = flipud(e_ilc);

for i = 1:length(e_ilc)
    z_inv(i) = fliplr(h(1:i))*we*e_ilc_inv(1:i);
end
z = flipud(z_inv');

coef_rm(:,iter+1) = coef_rm(:,iter)+epsi*z;
```

Listing A.4: PD code Robot Racer 7 - 1.4

```
%Q-filter
ws = 200;
Q = tf(1,[1/ws^2 sqrt(2)/ws 1]);
[Q_num,Q_den] = tfdata(Q,'v');

%first iteration
N = 7746;
Ts1 = 0.002;
Time1 = 0:Ts1:Ts1*(N-1);
coef_rm = zeros(N,6);
coef_rm_sim = timeseries(coef_rm,Time1);
save coef_rm_sim.mat -v7.3 coef_rm_sim

sim('test_ILC_q_PD.slx');

%cartesian error computation first iteration

output_q_motor_first_iter = output_q_motor;

xyz_in = zeros(3,N);
xyz_out{1} = zeros(3,N);
for i = 1:N

    xyz_in(1:3,i) = fkine_motor(RobotObject, input_q_motor(i,:)');
    xyz_out{1}(1:3,i) = fkine_motor(RobotObject, ...
        output_q_motor(i,:)');

end

xyz_err{1}(1:3,:) = xyz_in(1:3,:) - xyz_out{1}(1:3,:);

%optimal parameters
Kp = 0.33;
Kd = 3.5;
```

```
34 itermax = 50;
35 delta = 1;
36
37 %variable initialization
38 em_z = cell(itermax,1);
39 em_norm_inf = zeros(itermax,6);
40 em_norm_2 = zeros(itermax,6);
41 em_norm_1 = zeros(itermax,6);
42 RMSE_motor = zeros(itermax,6);
43 em_norm_inf_global = zeros(itermax,1);
44 em_norm_2_global = zeros(itermax,1);
45 em_norm_1_global = zeros(itermax,1);
46 RMSE_motor_global = zeros(itermax,1);
47
48 %other iteration
49 for iter = 1:itermax
50
51     iter
52
53     for i = 1:6
54
55         %data for 3D plot of em and el
56         em_z{i}(:,iter) = error_q_motor(:,i);
57
58         %norm1,2,Inf error for motor side
59         em_norm_1(iter,i) = norm(error_q_motor(:,i),1);
60         em_norm_2(iter,i) = norm(error_q_motor(:,i),2);
61         em_norm_inf(iter,i) = norm(error_q_motor(:,i),Inf);
62
63         %RMSE
64         RMSE_motor(iter,i) = sqrt(immse(output_q_motor_ref(:,i), ...
               ...
65                                         input_q_motor_ref(:,i)));
66
67     end
68
69     %global error
70     em_norm_inf_global(iter,1) = norm(error_q_motor,inf);
71     em_norm_2_global(iter,1) = norm(error_q_motor,2);
72     em_norm_1_global(iter,1) = norm(error_q_motor,1);
73     RMSE_motor_global(iter,1) = ...
           sqrt(immse(output_q_motor_ref,input_q_motor_ref));
74
75     if iter < itermax
76
77         %ILC
78
79         coef_rm = coef_rm_sim_f + Kp .* error_q_motor + Kd .* ...
80             ([error_q_motor(delta+1:end,:);zeros(delta,6)] - ...
81             error_q_motor);
```

```
82
83          coef_rm_sim = timeseries(coef_rm,Time1);
84          save coef_rm_sim.mat -v7.3 coef_rm_sim
85          sim('test_ILC_q_PD.slx');
86
87      else
88
89          %cartesian error computation last iteration
90
91          xyz_out{iter} = zeros(3,N);
92          for i = 1:N
93
94              xyz_out{iter}(1:3,i) = ...
                    fkine_motor(RobotObject,output_q_motor(i,:)');
95
96          end
97
98          xyz_err{iter}(1:3,:) = xyz_in(1:3,:) - ...
                xyz_out{iter}(1:3,:);
99
100         break
101
102     end
103 end
104
105 %% figure
106
107 iter_x = ones(N,1)*(1:iter);
108
109 %joint space plot
110 for i = 1:6
111
112     % normalization
113     normalized = 1;
114     if normalized
115         em_norm_inf(:,i) = em_norm_inf(:,i)/max(em_norm_inf(:,i));
116         em_norm_2(:,i) = em_norm_2(:,i)/max(em_norm_2(:,i));
117         em_norm_1(:,i) = em_norm_1(:,i)/max(em_norm_1(:,i));
118         RMSE_motor(:,i) = RMSE_motor(:,i)/max(RMSE_motor(:,i));
119     end
120
121     % plot
122
123     linear = 0;
124
125     if linear
126         fig1 = figure(1); subplot(3,2,i), ...
                plot(RMSE_motor(1:iter,i)), title('RMSE(rm-qm)'), ...
                grid on
127     else
```

```
128             fig1 = figure(1); subplot(3,2,i), ...
                    semilogy(RMSE_motor(1:iter,i)), title('RMSE(rm-qm)'), ...
                    grid on
129         end
130
131
132         fig2 = figure(2); subplot(3,2,i), plot3(iter_x, 1:1:N, ...
                em_z{i}), title('motor position error rm-qm'), grid on
133
134
135         fig3 = figure(3); subplot(3,2,i), plot(em_z{i}(:,1),'r'), ...
                hold on, plot(em_z{i}(:,iter),'b'), title(sprintf('motor ...
                position error rm-qm 1 iter, %d iter', iter)), grid on, ...
                legend('em\_init','em\_final')
136         fig4 = figure(4); subplot(3,2,i), ...
                plot(abs(em_z{i}(:,1)),'r'), hold on, ...
                plot(abs(em_z{i}(:,iter)),'b'), title(sprintf('abs motor ...
                position error |rm-qm| 1 iter, %d iter', iter)), grid on, ...
                legend('em\_init','em\_final')
137
138         fig5 = figure(5); subplot(3,2,i), ...
                plot(input_q_motor(:,i),'r','LineWidth',1), hold on, ...
                plot(output_q_motor_first_iter(:,i),'-.g','LineWidth',1), ...
                plot(output_q_motor(:,i),'--b','LineWidth',1), ...
                title('Reference motor vs qm'), grid on, ...
                legend('Ref\_m','qm\_initial','qm\_final')
139
140
141         if linear
142             fig6 = figure(6); subplot(3,2,i), ...
                    plot(em_norm_inf(1:iter,i)), title('infinity norm ...
                    em=rm-qm'), grid on
143             fig7 = figure(7); subplot(3,2,i), ...
                    plot(em_norm_2(1:iter,i)), title('2-norm em=rm-qm'), ...
                    grid on
144             fig8 = figure(8); subplot(3,2,i), ...
                    plot(em_norm_1(1:iter,i)), title('1-norm em=rm-qm'), ...
                    grid on
145         else
146             fig6 = figure(6); subplot(3,2,i), ...
                    semilogy(em_norm_inf(1:iter,i)), title('infinity norm ...
                    em=rm-qm'), grid on
147             fig7 = figure(7); subplot(3,2,i), ...
                    semilogy(em_norm_2(1:iter,i)), title('2-norm ...
                    em=rm-qm'), grid on
148             fig8 = figure(8); subplot(3,2,i), ...
                    semilogy(em_norm_1(1:iter,i)), title('1-norm ...
                    em=rm-qm'), grid on
149         end
150
```

```
151 end
152
153 %cartesian space plot
154 figure(9), subplot(3,1,1), plot(xyz_err{1}(1,:),'-r'), hold on, ...
        plot(xyz_err{iter}(1,:),'--b'), grid on, title('TCP error on ...
        X coordinate'), legend('First iteration','Last iteration'), ...
        xlabel('Points of the trajectory'), ylabel('X position error ...
        [m]')
155 figure(9), subplot(3,1,2), plot(xyz_err{1}(2,:),'-r'), hold on, ...
        plot(xyz_err{iter}(2,:),'--b'), grid on, title('TCP error on ...
        Y coordinate'), legend('First iteration','Last iteration'), ...
        xlabel('Points of the trajectory'), ylabel('Y position error ...
        [m]')
156 figure(9), subplot(3,1,3), plot(xyz_err{1}(3,:),'-r'), hold on, ...
        plot(xyz_err{iter}(3,:),'--b'), grid on, title('TCP error on ...
        Z coordinate'), legend('First iteration','Last iteration'), ...
        xlabel('Points of the trajectory'), ylabel('Z position error ...
        [m]')
157
158 figure(11), subplot(3,1,1), plot(xyz_in(1,:),'-r'), hold on, ...
        plot(xyz_out{iter}(1,:),'--b'), plot(xyz_out{1}(1,:),'--g'), ...
        title('error on X coord of TCP initial vs final'), grid on, ...
        legend('initial','final'), hold on, subplot(3,1,2), ...
        plot(xyz_in(2,:),'-r'), hold on, ...
        plot(xyz_out{iter}(2,:),'--b'), plot(xyz_out{1}(2,:),'--g'), ...
        title('error on Y coord of TCP initial vs final'), grid on, ...
        legend('initial','final'), subplot(3,1,3), ...
        plot(xyz_in(3,:),'-r'), hold on, ...
        plot(xyz_out{iter}(3,:),'--b'), plot(xyz_out{1}(3,:),'--g'), ...
        title('error on Z coord of TCP initial vs final'), grid on, ...
        legend('initial','final')
159
160 %3D plot
161 figure(12), plot3(xyz_in(1,:),xyz_in(2,:),xyz_in(3,:),'r'), ...
162     hold on, plot3(xyz_out{iter}(1,:),xyz_out{iter}(2,:), ...
163                    xyz_out{iter}(3,:),'--b'), ...
164     plot3(xyz_out{1}(1,:),xyz_out{1}(2,:),xyz_out{1}(3,:),'--g')
165
166 %global metric normalization
167 em_norm_inf_global(:,1) = ...
        em_norm_inf_global(:,1)/max(em_norm_inf_global(:,1));
168 em_norm_2_global(:,1) = ...
        em_norm_2_global(:,1)/max(em_norm_2_global(:,1));
169 em_norm_1_global(:,1) = ...
        em_norm_1_global(:,1)/max(em_norm_1_global(:,1));
170 RMSE_motor_global(:,1) = ...
        RMSE_motor_global(:,1)/max(RMSE_motor_global(:,1));
171
172 %global metric plot
173 if linear
```

```
174     figure(13), subplot(2,2,1), ...
            plot(em_norm_inf_global(1:iter,1)), title('Normalized ...
            infinity norm of motor side error'), grid on
175     figure(13), subplot(2,2,2), plot(em_norm_2_global(1:iter,1)), ...
            title('Normalized 2 norm of motor side error'), grid on
176     figure(13), subplot(2,2,3), plot(em_norm_1_global(1:iter,1)), ...
            title('Normalized 1 norm of motor side error'), grid on
177     figure(13), subplot(2,2,4), ...
            plot(RMSE_motor_global(1:iter,1)), title('Normalized RMSE ...
            of motor side error'), grid on
178 else
179     figure(13), subplot(2,2,1), ...
            semilogy(em_norm_inf_global(1:iter,1)), title('Infinity ...
            norm of motor side error'), grid on, xlabel('Iteration'), ...
            ylabel('Normalized inf-norm em')
180     figure(13), subplot(2,2,2), ...
            semilogy(em_norm_2_global(1:iter,1)), title('2-norm of ...
            motor side error'), grid on, xlabel('Iteration'), ...
            ylabel('Normalized 2-norm em')
181     figure(13), subplot(2,2,3), ...
            semilogy(em_norm_1_global(1:iter,1)), title('1-norm of ...
            motor side error'), grid on, xlabel('Iteration'), ...
            ylabel('Normalized 1-norm em')
182     figure(13), subplot(2,2,4), ...
            semilogy(RMSE_motor_global(1:iter,1)), title('RMSE of ...
            motor side error'), grid on, xlabel('Iteration'), ...
            ylabel('Normalized RMSE em')
183 end
```

Listing A.5: Data Driven code Robot Racer 7 - 1.4

```
1  load h.mat %Markov parameters
2  load epsilon_bar.mat  %max learning gain
3
4  %Q-filter
5  ws = 25;
6  Q = tf(1,[1/ws^2 sqrt(2)/ws 1]);
7  [Q_num,Q_den] = tfdata(Q,'v');
8
9  %first iteration
10 N = 7746;
11 Ts1 = 0.002;
12 Time1 = 0:Ts1:Ts1*(N-1);
13 coef_rm = zeros(N,6);
14 coef_rm_sim = timeseries(coef_rm,Time1);
15 save coef_rm_sim.mat -v7.3 coef_rm_sim
16
17 sim('test_ILC_q_DD.slx');
18
```

```
19 %cartesian error computation first iteration
20
21 output_q_motor_first_iter = output_q_motor;
22
23 xyz_in = zeros(3,N);
24 xyz_out{1} = zeros(3,N);
25 for i = 1:N
26
27     xyz_in(1:3,i) = fkine_motor(RobotObject, input_q_motor(i,:)');
28     xyz_out{1}(1:3,i) = fkine_motor(RobotObject, ...
           output_q_motor(i,:)');
29
30 end
31
32 xyz_err{1}(1:3,:) = xyz_in(1:3,:) - xyz_out{1}(1:3,:);
33
34 %iterations definition
35 itermax = 50;
36 delta = 1;
37
38 %variable initialization
39 coef_rm = cell(itermax,1);
40 coef_rm{1} = zeros(N,6);
41 coef_rm_selected = zeros(N,6);
42 em_z = cell(itermax,1);
43 em_norm_inf = zeros(itermax,6);
44 em_norm_2 = zeros(itermax,6);
45 em_norm_1 = zeros(itermax,6);
46 RMSE_motor = zeros(itermax,6);
47 em_norm_inf_global = zeros(itermax,1);
48 em_norm_2_global = zeros(itermax,1);
49 em_norm_1_global = zeros(itermax,1);
50 RMSE_motor_global = zeros(itermax,1);
51
52 %other iteration
53 for iter = 1:itermax
54
55     iter
56
57     for i = 1:6
58
59         %data for 3D plot of em and el
60         em_z{i}(:,iter) = error_q_motor(:,i);
61
62         %norm1,2,Inf error for motor side
63         em_norm_1(iter,i) = norm(error_q_motor(:,i),1);
64         em_norm_2(iter,i) = norm(error_q_motor(:,i),2);
65         em_norm_inf(iter,i) = norm(error_q_motor(:,i),Inf);
66
67         %RMSE
```

```
68          RMSE_motor(iter,i) = sqrt(immse(output_q_motor_ref(:,i), ...
                ...
69                                      input_q_motor_ref(:,i)));
70
71      end
72
73      %global error
74      em_norm_inf_global(iter,1) = norm(error_q_motor,inf);
75      em_norm_2_global(iter,1) = norm(error_q_motor,2);
76      em_norm_1_global(iter,1) = norm(error_q_motor,1);
77      RMSE_motor_global(iter,1) = ...
            sqrt(immse(output_q_motor_ref,input_q_motor_ref));
78
79      if iter < itermax
80
81      %DD-ILC
82
83          e_reverse = zeros(1,N);
84          z_reverse = zeros(1,N);
85          z_not_reverse = zeros(N,6);
86          for j = 1:6
87
88                  e_reverse = fliplr(error_q_motor(:,j)');
89
90                  for jj = 1:6
91
92                      for i = 1:N
93                          z_reverse(1,i) = ...
                                fliplr(h{j,jj}(1,1:i))*e_reverse(1,1:i)';
94                      end
95
96                      z_not_reverse(:,j) = z_not_reverse(:,j) + ...
                            flipud(z_reverse');
97
98                  end
99
100         end
101
102         epsilon = 0.5 * epsilon_bar;
103         coef_rm{iter+1} = coef_rm{iter} + epsilon .* z_not_reverse;
104         coef_rm_sim = timeseries(coef_rm{iter+1},Time1);
105         save coef_rm_sim.mat -v7.3 coef_rm_sim
106
107         sim('test_ILC_q_DD.slx');
108
109     else
110
111     %cartesian error computation last iteration
112
113         xyz_out{iter} = zeros(3,N);
```

```
114             for i = 1:N
115
116                 xyz_out{iter}(1:3,i) = ...
                        fkine_motor(RobotObject,output_q_motor(i,:)');
117
118             end
119
120             xyz_err{iter}(1:3,:) = xyz_in(1:3,:) - ...
                    xyz_out{iter}(1:3,:);
121             break
122
123         end
124     end
125
126     %% figure
127
128     iter_x = ones(N,1)*(1:iter);
129
130     %joint space plot
131     for i = 1:6
132
133         % normalization
134         normalized = 1;
135         if normalized
136             em_norm_inf(:,i) = em_norm_inf(:,i)/max(em_norm_inf(:,i));
137             em_norm_2(:,i) = em_norm_2(:,i)/max(em_norm_2(:,i));
138             em_norm_1(:,i) = em_norm_1(:,i)/max(em_norm_1(:,i));
139             RMSE_motor(:,i) = RMSE_motor(:,i)/max(RMSE_motor(:,i));
140         end
141
142         % plot
143
144         linear = 0;
145
146         if linear
147             fig1 = figure(1); subplot(3,2,i), ...
                        plot(RMSE_motor(1:iter,i)), title('RMSE(rm-qm)'), ...
                        grid on
148         else
149             fig1 = figure(1); subplot(3,2,i), ...
                        semilogy(RMSE_motor(1:iter,i)), title('RMSE(rm-qm)'), ...
                        grid on
150         end
151
152
153         fig2 = figure(2); subplot(3,2,i), plot3(iter_x, 1:1:N, ...
                em_z{i}), title('motor position error rm-qm'), grid on
154
155
```

```
156     fig3 = figure(3); subplot(3,2,i), plot(em_z{i}(:,1),'r'), ...
            hold on, plot(em_z{i}(:,iter),'b'), title(sprintf('motor ...
            position error rm-qm 1 iter, %d iter', iter)), grid on, ...
            legend('em\_init','em\_final')
157     fig4 = figure(4); subplot(3,2,i), ...
            plot(abs(em_z{i}(:,1)),'r'), hold on, ...
            plot(abs(em_z{i}(:,iter)),'b'), title(sprintf('abs motor ...
            position error |rm-qm| 1 iter, %d iter', iter)), grid on, ...
            legend('em\_init','em\_final')
158
159     fig5 = figure(5); subplot(3,2,i), ...
            plot(input_q_motor(:,i),'r','LineWidth',1), hold on, ...
            plot(output_q_motor_first_iter(:,i),'-.g','LineWidth',1), ...
            plot(output_q_motor(:,i),'--b','LineWidth',1), ...
            title('Reference motor vs qm'), grid on, ...
            legend('Ref\_m','qm\_initial','qm\_final')
160
161
162     if linear
163         fig6 = figure(6); subplot(3,2,i), ...
                plot(em_norm_inf(1:iter,i)), title('infinity norm ...
                em=rm-qm'), grid on
164         fig7 = figure(7); subplot(3,2,i), ...
                plot(em_norm_2(1:iter,i)), title('2-norm em=rm-qm'), ...
                grid on
165         fig8 = figure(8); subplot(3,2,i), ...
                plot(em_norm_1(1:iter,i)), title('1-norm em=rm-qm'), ...
                grid on
166     else
167         fig6 = figure(6); subplot(3,2,i), ...
                semilogy(em_norm_inf(1:iter,i)), title('infinity norm ...
                em=rm-qm'), grid on
168         fig7 = figure(7); subplot(3,2,i), ...
                semilogy(em_norm_2(1:iter,i)), title('2-norm ...
                em=rm-qm'), grid on
169         fig8 = figure(8); subplot(3,2,i), ...
                semilogy(em_norm_1(1:iter,i)), title('1-norm ...
                em=rm-qm'), grid on
170     end
171
172 end
173
174 %cartesian space plot
175 figure(9), subplot(3,1,1), plot(xyz_err{1}(1,:),'-r'), hold on, ...
        plot(xyz_err{iter}(1,:),'--b'), grid on, title('TCP error on ...
        X coordinate'), legend('First iteration','Last iteration'), ...
        xlabel('Points of the trajectory'), ylabel('X position error ...
        [m]')
```

```
176 figure(9), subplot(3,1,2), plot(xyz_err{1}(2,:),'-r'), hold on, ...
        plot(xyz_err{iter}(2,:),'--b'), grid on, title('TCP error on ...
        Y coordinate'), legend('First iteration','Last iteration'), ...
        xlabel('Points of the trajectory'), ylabel('Y position error ...
        [m]')
177 figure(9), subplot(3,1,3), plot(xyz_err{1}(3,:),'-r'), hold on, ...
        plot(xyz_err{iter}(3,:),'--b'), grid on, title('TCP error on ...
        Z coordinate'), legend('First iteration','Last iteration'), ...
        xlabel('Points of the trajectory'), ylabel('Z position error ...
        [m]')
178
179 figure(11), subplot(3,1,1), plot(xyz_in(1,:),'-r'), hold on, ...
        plot(xyz_out{iter}(1,:),'--b'), plot(xyz_out{1}(1,:),'--g'), ...
        title('error on X coord of TCP initial vs final'), grid on, ...
        legend('initial','final'), hold on, subplot(3,1,2), ...
        plot(xyz_in(2,:),'-r'), hold on, ...
        plot(xyz_out{iter}(2,:),'--b'), plot(xyz_out{1}(2,:),'--g'), ...
        title('error on Y coord of TCP initial vs final'), grid on, ...
        legend('initial','final'), subplot(3,1,3), ...
        plot(xyz_in(3,:),'-r'), hold on, ...
        plot(xyz_out{iter}(3,:),'--b'), plot(xyz_out{1}(3,:),'--g'), ...
        title('error on Z coord of TCP initial vs final'), grid on, ...
        legend('initial','final')
180
181 %3D plot
182 figure(12), plot3(xyz_in(1,:),xyz_in(2,:),xyz_in(3,:),'r'), ...
183     hold on, plot3(xyz_out{iter}(1,:),xyz_out{iter}(2,:), ...
184                    xyz_out{iter}(3,:),'--b'), ...
185     plot3(xyz_out{1}(1,:),xyz_out{1}(2,:),xyz_out{1}(3,:),'--g')
186
187 %global metric normalization
188 em_norm_inf_global(:,1) = ...
        em_norm_inf_global(:,1)/max(em_norm_inf_global(:,1));
189 em_norm_2_global(:,1) = ...
        em_norm_2_global(:,1)/max(em_norm_2_global(:,1));
190 em_norm_1_global(:,1) = ...
        em_norm_1_global(:,1)/max(em_norm_1_global(:,1));
191 RMSE_motor_global(:,1) = ...
        RMSE_motor_global(:,1)/max(RMSE_motor_global(:,1));
192
193 %global metric plot
194 if linear
195     figure(13), subplot(2,2,1), ...
          plot(em_norm_inf_global(1:iter,1)), title('Normalized ...
          infinity norm of motor side error'), grid on
196     figure(13), subplot(2,2,2), plot(em_norm_2_global(1:iter,1)), ...
          title('Normalized 2 norm of motor side error'), grid on
197     figure(13), subplot(2,2,3), plot(em_norm_1_global(1:iter,1)), ...
          title('Normalized 1 norm of motor side error'), grid on
```

```
198         figure(13), subplot(2,2,4), ...
                plot(RMSE_motor_global(1:iter,1)), title('Normalized RMSE ...
                of motor side error'), grid on
199     else
200         figure(13), subplot(2,2,1), ...
                semilogy(em_norm_inf_global(1:iter,1)), title('Infinity ...
                norm of motor side error'), grid on, xlabel('Iteration'), ...
                ylabel('Normalized inf-norm em')
201         figure(13), subplot(2,2,2), ...
                semilogy(em_norm_2_global(1:iter,1)), title('2-norm of ...
                motor side error'), grid on, xlabel('Iteration'), ...
                ylabel('Normalized 2-norm em')
202         figure(13), subplot(2,2,3), ...
                semilogy(em_norm_1_global(1:iter,1)), title('1-norm of ...
                motor side error'), grid on, xlabel('Iteration'), ...
                ylabel('Normalized 1-norm em')
203         figure(13), subplot(2,2,4), ...
                semilogy(RMSE_motor_global(1:iter,1)), title('RMSE of ...
                motor side error'), grid on, xlabel('Iteration'), ...
                ylabel('Normalized RMSE em')
204     end
```

Appendix B

Simulink Scheme

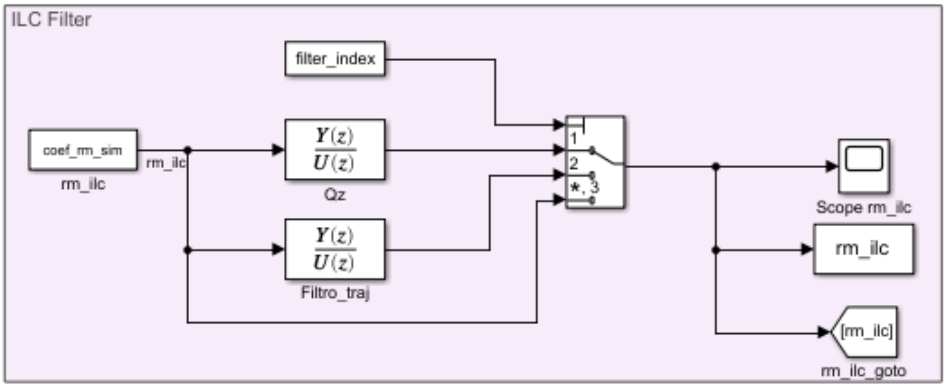

Figure B.1: ILC filter of the command to be applied.

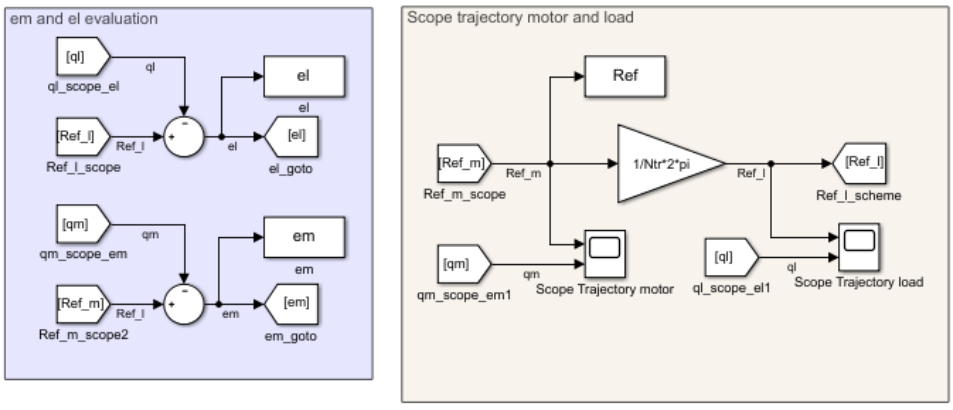

Figure B.2: Evaluation of motor and load variables.

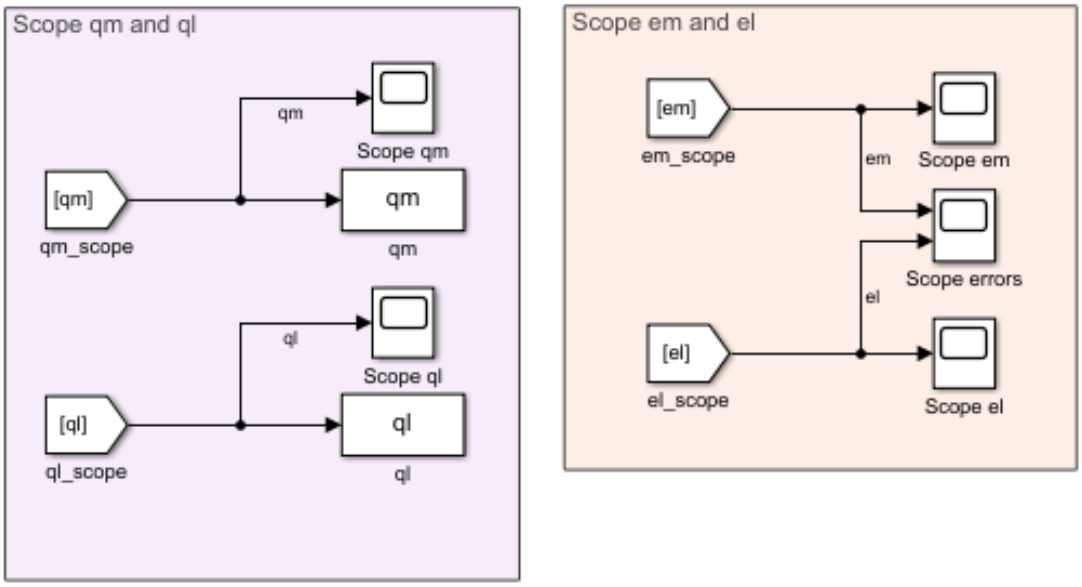

Figure B.3: Scope of motor and load variables.

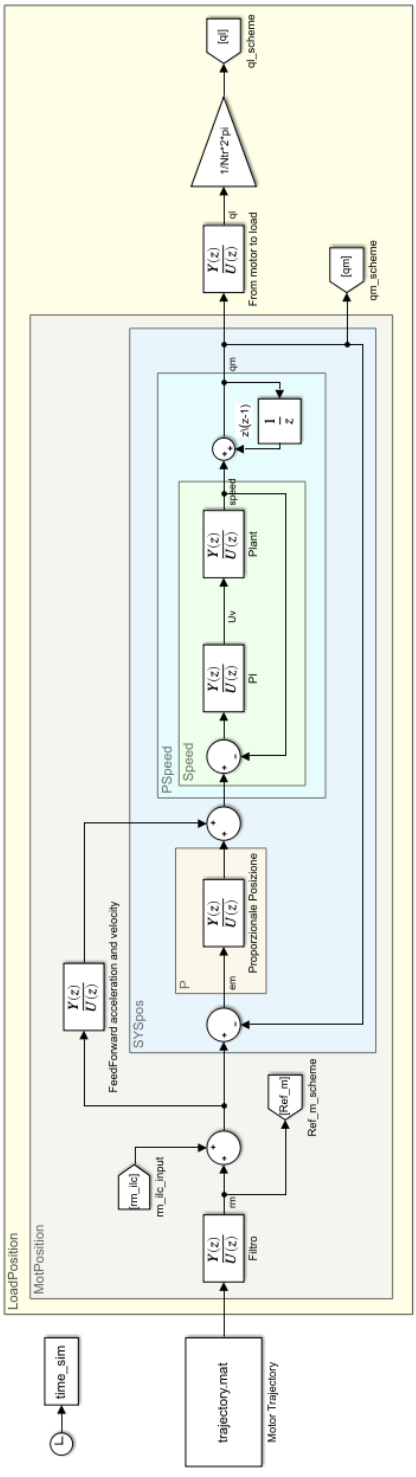

Figure B.4: Simplified SISO system of a 4^{th} link for the Comau S.p.A. Robot NJ4 220 - 2.4.

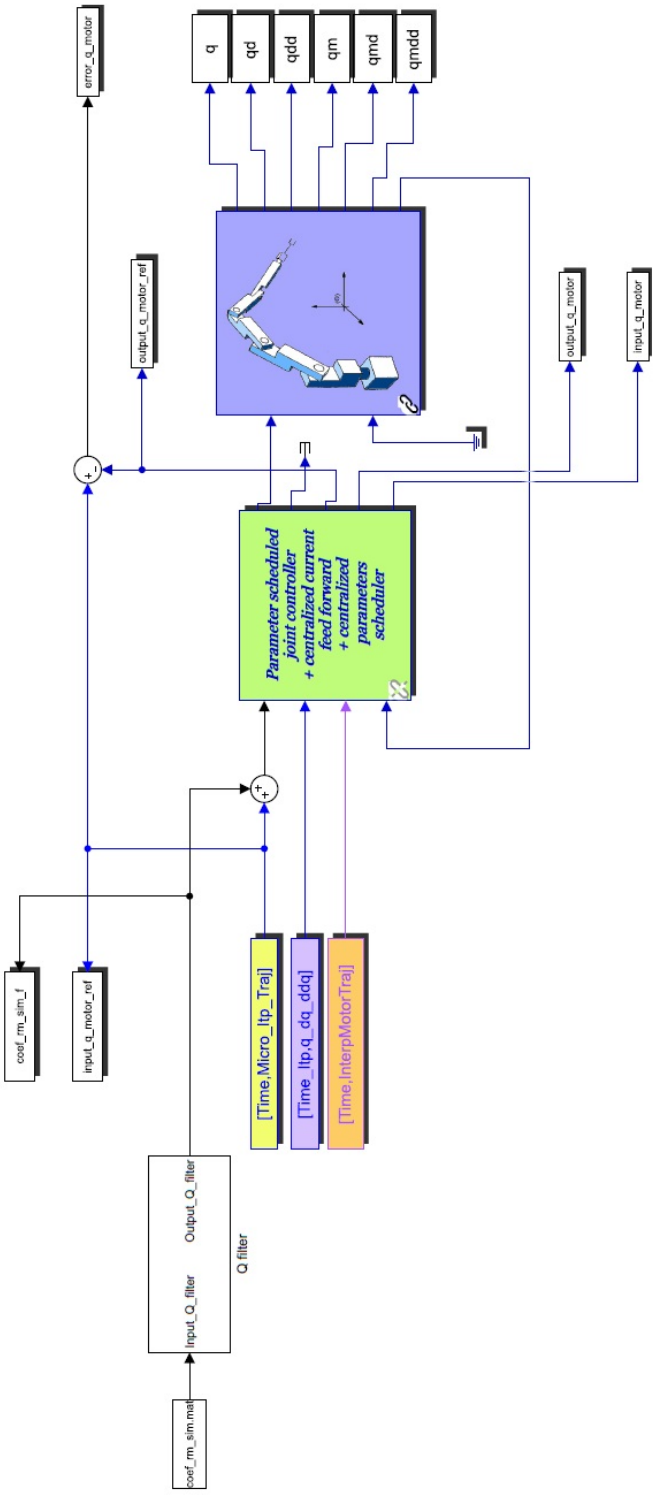

Figure B.5: MIMO system for the Comau S.p.A. Robot Racer 7 - 1.4.

Bibliography

[1] S. Arimoto, S. Kawamura, and F. Miyazaki, "*Bettering operation of robots by learning*", J. Robot. Syst., vol. 1, pp. 123–140, 1984.

[2] G. Casalino and G. Bartolini, A learning procedure for the control of movements of robotic manipulators, In IASTED Sym. Robot. Autom., pages 108–111, San Francisco, USA, May 1984.

[3] J.J. Craig., "*Adaptive control of manipulators through repeated trials*", In Proc. American Control Conf., pages 1566–1572, San Diego, CA, June 1984.

[4] M. Uchiyama, "*Formulation of high-speed motion pattern of a mechanical arm by trial*", Trans. Soc. Instrum. Control Eng., vol. 14, no. 6, 1978.

[5] R. Middleton, G.C. Goodwin and R.W. Longman, "*A method for improving the dynamic accuracy of a robot performing a repetitive task*", Technical Report EE8546, Dept. Electrical Engineering, Univ. of Newcastle, Australia, 1985.

[6] D.A. Bristow, M. Tharayil, A.G. Alleyne, "*A Survey Of Iterative Learning Control: A Learning-Based Method for High-Performance Tracking Control*", IEEE Control Systems, vol.3, pp. 96-114, 2006.

[7] K.L. Moore, "*Iterative Learning Control for Deterministic Systems*", London: Springer-Verlag, 1993.

[8] K.J. Hunt, D. Sbarbaro, R. Zbikowski, and P.J. Gawthrop, "*Neural networks for control systems-A survey*", Automatica, vol. 28, no. 6, pp. 1083–112, 1992.

[9] R.W. Longman, "*Iterative learning control and repetitive control for engineering practice*", Int. J. Contr., vol. 73, no. 10, pp. 930–954, 2000.

[10] R. Longman, "*Designing iterative learning and repetitive controllers, in Iterative Learning Control: Analysis, Design, Integration and Applications*", Z. Bien and J.-X. Xu, Eds. Boston: Kluwer, 1998.

[11] D. de Roover and O.H. Bosgra, "*Synthesis of robust multivariable iterative learning controllers with application to a wafer stage motion system*", Int. J. Contr., vol. 73, no. 10, pp. 968–979, 2000.

[12] H. Havlicsek and A. Alleyne, "*Nonlinear control of an electrohydraulic injection molding machine via iterative adaptive learning*", IEEE/ASME Trans. Mechatron., vol. 4, no. 3, pp. 312–323, 1999.

[13] K. Kinosita, T. Sogo, and N. Adachi, "*Iterative learning control using adjoint systems and stable inversion*", Asian J. Contr., vol. 4, no. 1, pp. 60–67, 2002.

[14] T. Sogo, "*Stable inversion for nonminimum phase sampled-data systems and its relation with the continuous-time counterpart*", in Proc. 41st IEEE Conf. Decision Contr., 2002, pp. 3730–3735.

[15] D. de Roover, "*Synthesis of a robust iterative learning controller using an H*inf *approach*", in Proc. 35th IEEE Conf. Decision Contr., 1996, pp. 3044–3049.

[16] C.J. Goh and W.Y. Yan, "*An H*inf *synthesis of robust current error feedback learning control*", J. Dyn. Syst. Meas. Control, vol. 118, no. 2, pp. 341–346, 1996.

[17] N. Amann, D.H. Owens, and E. Rogers, "*Iterative learning control for discrete-time systems with exponential rate of convergence*", IEE Proc.: Control Theory Applicat., vol. 143, no. 2, pp. 217–224, 1996.

[18] J. Bolder, T. Oomen, "*Data-driven optimal ILC for multivariable systems: Removing the need for L and Q filter design*", American Control Conference (ACC), DOI 10.1109/ACC.2015.7171880, 2015.

[19] C. Wang, M. Zheng, Z. Wang, M. Tomizuka, "*Robust two-degree-of-freedom iterative learning control for flexibility compensation of industrial robot manipulator*", Proc. International Conference on Robotics and Automation (ICRA), pp. 2381-2386, 2016.

[20] M. Norrlof and S. Gunnarsson, "*Time and frequency domain convergence properties in iterative learning control*", Int. J. Contr., vol. 75, no. 14, pp. 1114–1126, 2002.

[21] C.T. Chen, "*Linear System Theory and Design*", New York: Oxford Univ. Press, 1999.

[22] U. Grenander and G. Szego, "*Toeplitz Forms and their Applications*", Berkeley, CA: Univ. of California Press, 1958.

[23] M.Q. Phan, R.W. Longman, and K.L. Moore, "*Unified formulation of linear iterative learning control*", Adv. Astronautical Sci., vol. 105, pp. 93–111, 2000.

[24] N. Amann, D.H. Owens, E. Rogers, and A. Wahl, "*An H∞ approach to linear iterative learning control design*", Int. J. Adaptive Contr. Signal Processing, vol. 10, no. 6, pp. 767–781, 1996.

[25] Y. Chen, Z. Gong, and C. Wen, "*Analysis of a high-order iterative learning control algorithm for uncertain nonlinear systems with state delays*", Automatica, vol. 34, no. 3, pp. 345–353, 1998.

[26] S.A. Saab, "*A stochastic iterative learning control algorithm with application to an induction motor*", Int. J. Contr., vol. 77, no. 2, pp. 144–163, 2004.

[27] H.S. Lee and Z. Bien, "*Study on robustness of iterative learning control with non-zero initial error*", Int. J. Contr., vol. 64, no. 3, pp. 345–359, 1996.

[28] R. Horowitz, "*Learning control of robot manipulators*", Trans. ASME J. Dyn. Syst. Meas. Control, vol. 115, no. 2B, pp. 402–411, 1993.

[29] S.S. Saab, "*Stochastic P-type/D-type iterative learning control algorithms*", Int. J. Contr., vol. 76, no. 2, pp. 139–148, 2003.

[30] K.H. Park, Z. Bien, and D.H. Hwang, "*Study on the robustness of a PID-type iterative learning controller against initial state error*", Int. J. Syst. Sci., vol. 30, no. 1, pp. 49–59, 1999.

[31] Y. Chen and K.L. Moore, "*An optimal design of PD-type iterative learning control*

with monotonic convergence", in Proc. IEEE Int. Symp. Intelligent Contr., 2002, pp. 55–60.

[32] H. Elci, R.W. Longman, M. Phan, J.N. Juang, and R. Ugoletti, "*Discrete frequency based learning control for precision motion control*", in Proc. IEEE Int. Conf. Syst., Man, Cybern., 1994, pp. 2767–2773.

[33] T. Kavli, "*Frequency domain synthesis of trajectory learning controllers for robot manipulators*", J. Robot. Syst., vol. 9, no. 5, pp. 663–680, 1992.

[34] D.A. Bristow and A.G. Alleyne, "*A manufacturing system for microscale robotic deposition*", in Proc. Ame. Contr. Conf., 2003, pp. 2620–2625.

[35] D.I. Kim and S. Kim, "*An iterative learning control method with application for CNC machine tools*", IEEE Trans. Ind. Applicat., vol. 32, no. 1, pp. 66–72, 1996.

[36] A. Naylor and G. Sell, "*Linear Operator Theory in Engineering and Science*", Springer, 1982.

[37] Tomizuka M., "*Zero Phase Error Tracking Algorithm for Digital Control*", ASME. J. Dyn. Sys., Meas., Control.;109(1):65-68. doi:10.1115/1.3143822, 1987.