Politecnico di Milano
School of Industrial and Information Engineering
Master of Science in Aeronautical Engineering

A norm-optimal Kalman iterative learning control for precise UAV trajectory tracking

Advisor:	Prof. Marco LOVERA
Co-Advisor:	Eng. Salvatore MERAGLIA

Thesis by:
Francesco CARLONI Matr. 905340

Academic Year 2019–2020

Alla mia Famiglia

Acknowledgments

Desidero dedicare questo spazio a tutti coloro che hanno contribuito, direttamente o indirettamente, alla realizzazione del mio elaborato.
Un sentito ringraziamento va al Professor Marco Lovera per avermi dato l'opportunità di svolgere questa tesi, per la professionalità e disponibilità mostrata.
Una menzione speciale la rivolgo anche al mio correlatore Ing. Salvatore Meraglia che nonostante il contatto a distanza, data la situazione contingente, è stato sempre presente con indicazioni e suggerimenti e ha portato a termine l'attività sperimentale.
Grazie alla mia famiglia per aver creduto in me e per avermi sempre messo nelle condizioni migliori per raggiungere questo obiettivo.
Infine desidero ringraziare parenti e amici per avermi supportato e aiutato nell'affrontare questo lungo e impegnativo percorso.

Abstract

The interest in multirotor Unmanned Aerial Vehicles (UAVs) has faced an exponential growth over the last decades. The technological success, the cost reduction and the parallel diffusion led to the adoption of UAVs in many research fields. Despite their massive popularity, the algorithms deployed to control UAVs typically rely on simplified models approximating the complex flight dynamics. This can lead to significant tracking errors when performing aggressive manoeuvres. Iterative learning control (ILC) algorithms can be used in combination with the traditional feedback control methods, which are generally implemented to correct the system behaviour by compensating for noise and unexpected disturbances as they occur, to reject recurring tracking errors happening during repetitive executions. These algorithms exploit the information acquired from past trials to shape non-causal feed-forward input signals so as to anticipate recurring disturbances and proactively compensate for them.

The purpose of this thesis is to design an iterative learning control algorithm for precise UAV trajectory tracking. This work focuses specifically in a new model-based, norm-optimal ILC, enhanced with a iteration-domain Kalman filter disturbance estimator.

The so called Kalman Iterative Learning Control (K-ILC) is implemented and validated in a simulation environment. Eventually, the effectiveness of the algorithm is successfully tested through experimental activity in a real quadcopter flying a eight-shape manoeuvrer.

Sommario

L'interesse verso gli aeromobili a pilotaggio remoto, comunemente noti come droni, ha subito una crescita esponenziale negli ultimi decenni. Il successo tecnologico, la riduzione dei costi e la parallela diffusione hanno portato all'adozione di tali velivoli in numerosi ambiti di ricerca. Nonostante la loro enorme popolarità, gli algoritmi tipicamente impiegati per controllare tali velivoli si basano su modelli semplificati che approssimano la complessa dinamica di volo. Ciò può portare a significativi errori di tracciamento durante l'esecuzione di manovre complesse ad alta velocità. Gli algoritmi di controllo di apprendimento iterativo (Iterative Learning Controls o ILCs) possono essere utilizzati in combinazione con i sistemi di controllo tradizionali in retroazione che compensano rumori e disturbi imprevisti. Questi, infatti, agiscono in modo da eliminare gli errori di posizionamento che si ripetono durante l'esecuzione della stessa manovra. Le informazioni acquisite dalle precedenti prove vengono sfruttate per calcolare segnali di ingresso non causali in modo da anticipare i disturbi ricorrenti e compensarli in modo proattivo.

Lo scopo di questa tesi è progettare un algoritmo di controllo di apprendimento iterativo per l'inseguimento della traiettoria di un drone. Questo lavoro si concentra in particolare su un nuovo ILC basato su un modello ottimale di aggiornamento del segnale in ingresso potenziato con un filtro di Kalman nel dominio dell'iterazione per migliorare la stima del disturbo.

Il cosiddetto Kalman Iterative Learning Control (K-ILC) è stato implementato e validato in simulazione. Infine, l'efficacia dell'algoritmo è stata verificata con successo attraverso l'attività sperimentale condotta nell'arena FlyART del laboratorio di controllo e sistemi aerospaziali (ASCL) del Politecnico di Milano.

Contents

List of Figures

List of Tables

Introduction

Unmanned Aerial Vehicles (UAVs), commonly called drones, are aircraft with no on-board crew or passengers, able to fly either under remote control or autonomously. With the maturing and miniaturization of applicable technologies, interest in UAVs grew at first within the military industry. Initial applications primarily involved combat surveillance, tactical reconnaissance and targeted attacks. In recent years, drones has receive increasing attention also in a wide range of civil applications. Examples include non-trivial inspection tasks, environmental monitoring, imagining for photogrammetry, search operations (*e.g.*, after natural disasters), photography and filming, farming operations, goods delivery, etc. In all these applications and in many others, high tracking precision is often required (even when performing aggressive manoeuvres) as it determines the quality of the task carried out. This thesis focuses specifically on multirotor Vertical Take-Off and Landing (VTOL) vehicles of small and medium size autonomously controlled by on-board computers referred to as autopilots.

These UAVs are capable of performing complex manoeuvres. The control laws which regulate their motions are commonly based on simple models, generally consisting of first-order approximations, which well describe the dynamics of multirotors during near-hover conditions. The model-based feedback controllers ensure the tracking of reference trajectories also when unmodeled effects are significant. In fact, they are intended to compensate for effects not captured by the nominal model (including noises and undesired disturbances). These effects include for example external disturbances, as wind and gusts, ground and wall interferences, highly non-linear aerodynamic loads due to the interaction of the vehicle when moving towards the turbulent wakes of propellers and lift and drag variations of rotary wings under unsteady inflow conditions.

In high-performing manoeuvres, the dynamic behaviour of the system under study is difficult to identify as those secondary effects are not negligible. As a consequence, the feedback is not able to react in time, causing a degradation of the tracking performance. When the same motion is executed repeatedly, the same errors due to unmodeled dynamics, parameter uncertainties, and other repeating disturbances, recur. The problem of rejecting these repetitive errors has already been addressed in motion control applications for industrial robotics, in which high-precision in tracking reference signals is crucial. An effective solution is the use of the so called Iterative Learning Control methods. The idea behind these approaches is to exploit the information acquired in past trials to improve the tracking performance of the system by adjusting the input signals of the following iterations. These learning algorithms have proved to be effective in compensating for repetitive errors when combined with traditional feedback control loops to correct non-repetitive disturbances as they occurs.

Iterative learning control algorithms have only recently been applied to improve the accuracy of autonomous UAVs in tracking predefined trajectories. ILC schemes are applicable in drones used for operations in which repetition is inherent to the required task and where the path followed by the drone is predefined. ILC autopilots could be employed for instance in farming (*e.g.*, for spraying pesticides on crops), environmental monitoring (of crops, forests, rivers, etc.), civil instruction inspections (*e.g.*, power lines, pipelines, dimes, highways, bridges) and filming (of a pre-defined scene with a camera mounted on the drone).

State of art

In the literature, the problem of controlling autonomous multirotor VTOL-UAVs for precise trajectory tracking (of aggressive flying) is a commonly addressed problem in control theory as multirotors are unstable non-linear systems with complex behaviour, especially at high speed.

The first methods developed extend the classical approaches with adaptation in order to cope with important model uncertainties and external disturbances. PID schemes, feedback linerized controllers and backstepping methods fall in the category of causal controllers as they are not indented to improve the system performance in executing a same trajectory repeatedly by learning from past iterations.

Other control strategies are based on reinforcement learning techniques (such as neural networks and output feedback to learn the complete dynamics of the UAV online) or adaptive control strategies (to adjust the control parameters to ensure high-robustness to disturbances).

The application of learning algorithms to flying vehicles, specifically the non-causal strategies designed to exploit past experience in order to improve future executions, is more rare and has been actively researched during recent years.

The problem of learning parametrized motions has been addressed in Lupashin and D'Andrea [1]. The trajectories are described by a set of design parameters that are adjusted after the execution of the manoeuvres to compensate for disturbances. An adaptation strategy to correct for trajectory periodic errors of generic repeated motions is carried out in the *frequency*-domain (with a Fourier series decomposition of the input and output signals) in [2]. Purwin and D'Andrea in [3] introduced a least-squares-based ILC, considering a *lifted*-domain description of the model under study, to non-causally anticipate recurring disturbances when driving a quadrotor quickly from a state to another.

Schoellig, Mueller and D'Andrea [4] [5] implemented and validated through experimental activity an optimization-based iterative learning for precise quadcopter trajectory tracking. The proposed algorithm combines traditional optimal filtering methods with state-of-the art optimization techniques in order to obtain an effective and computationally efficient learning strategy, that updates the feed-forward input signal according to a customizable learning objective.

In Degen and Schoellig [6] the advantages of Kalman-filter-enhanced ILC algorithms are highlighted. The so called Kalman-ILC has proven to be an effective method for improving the performance of repetitive control tasks of a single-input, single-output mass-spring-damper system.

Objective

The objective of this thesis is to develop and validate a novel ILC for precise trajectory tracking of an autonomous VTOL-UAV performing a periodic motion. ILC algorithms are, therefore, illustrated and discussed. The emphasis is put on *discrete*-time algorithms due to the digital nature of the computers involved in controlling UAVs and storing data acquired during trials. The approach selected is based on a so defined *lifted*-domain description of the model capturing the system's key dynamics.

The 'standard' optimization-based (also known as norm-optimal) ILC is enhanced with a Kalamn filter estimator to achieve both fast initial convergence and good noise rejection by estimating the repetitive disturbances and optimally adapting the learning update rule over the trials. The so called Kalamn Iterative Learning control (K-ILC) is detailed and analysed. In contrast to the benchmark problem presented in [6], this thesis addresses the challenge of designing a Kalman-ILC to a multirotor UAV for precisely tracking an arbitrary three-dimensional trajectory.

The main contribution of this thesis is the implementation and validation of the K-ILC to a complete pre-existing autonomous quadrotor model, for the execution of an aggressive manoeuvre, without simplifying or altering the inner dynamics of the system. The effectiveness of the iteartive control and its performance are investigated through experimental activity for a quadrotor.

Structure of the thesis

The thesis is organized as follows:

- in Chapter 1, the Iterating Learning Control is introduced; a general description of the control is given and an overview ILC algorithms is presented. Then, two different system representations are described, with emphasis on the lifted form representation of discrete-time linear systems, followed by a performance analysis discussion. Eventually, the typical design techniques are proposed with a focus towards quadratically norm-optimal approaches.

- Chapter 2 is centred on the main topic of this thesis: the Kalman iterative learning control. First, an introduction to the reasons of K-ILC design choice is discussed, then, the system description and its lifted-domain representation are presented. Later on, the Kalman-ILC algorithm is outlined and the estimation step (Kalamn filter) and input update step (optimization problem) are explained in detail. The chapter ends with an analysis of the design parameter choices and a comparison between the K-ILC and the Q-ILC.

- Chapter 3 contains the description of the Simulink model and the MATLAB code developed to test the performance of a Kalman iterative learning control for a quadrotor Unmanned Air Vehicle (UAV). An overview of the design process in terms of the general architecture and the implementation steps followed is illustrated. The K-ILC is first implemented in a general feedback SISO system and then applied to a MIMO system. Once the effectiveness of the control action is validated, the iterative control is installed in a pre-existent quadcopter model. The Simulink K-ILC quadcopter model and the MATLAB code structure of the simulation are detailed. Eventually, the quadrotor simulation results are presented and the choice of the main design parameters used is discussed.

- In Chapter 4 the trajectory tracking performance of the norm-optimal Kalman iterative learning control algorithm is tested in a real drone manoeuvre. First, an overview of the experimental setup (hardware and software) is illustrated. Then, the flight tests executed are described. Finally the experimental results are presented and discussed.

Chapter 1

Iterative Learning Control

In this chapter, the Iterating Learning Control, ILC for short, is introduced. A general description of the control is given and an overview ILC algorithms is presented. Then, two different system representations are described, with emphasis on the lifted form representation of discrete-time linear systems, followed by a performance analysis discussion. Eventually, the typical design techniques are proposed; the focus is towards quadratically norm-optimal approaches.

1.1 Introduction to ILC

Iterative Learning Control is a control strategy based on the idea that the performance of a dynamic system that executes the same task multiple times can be improved by introducing a correction signal to the system, dependent on the way in which the task was performed in previous iterations (trials).

ILC is a *learning*-type control (that is where the term *learning* comes from), since it uses information gathered during previous executions to acquire new understanding of the system. This knowledge allows the control to modify the system performance through a proper action on the signals that enter the system.

The term *iterative* refers, instead, to the repetitive nature of the learning process: the control, in fact, is implemented to systems that perform the same task repeatedly and under the same operating conditions.

1.1.1 Main idea

The idea of ILC is inspired by human learning and has its origin in industrial robot applications where a specific task is performed repeatedly under the same operating conditions and where high precision is vital.

Take, for example, a basketball player shooting a free throw from a fixed position; he or she can improve his or her ability to score by practising 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 [7].

ILC is exactly an open-loop control strategy that generates the control signal through practice.

This control method exploits every possibility to incorporate past control information, such as tracking errors and control input signals, into the construction of the next control action. This is done in two separate steps: first the long term memory components are used to store control information during trials, then the stored control information is retrieved and fused to update the input signal for the next iteration.

Due to its simplicity and effectiveness, ILC has received considerable attention in many fields of application.

1.1.2 Historical background

The term iterative learning control was introduced by Arimoto in 1984 and has become the standard notion. Actually, the first academic contribution to what today is called ILC is a paper by Uchiyama published in 1978; but, since it was only published in Japanese, the idea was not widely spread until 1984, when the papers by Arimoto *et al* [8]., Casalino and Bartolini [9], and Craig [10] were independently published.

The development of ILC grew originally from practical issues in the field of industrial robotics, where repetitive motions appear in a multitude of applications. Since then, ILC started to become an active research area. Much of the early work focuses on convergence of ILC algorithms and robustness and performance analysis.

Algorithms of different nature (both linear and nonlinear) for different system descriptions have been developed and analysed, but a unifying theory of ILC is still under development. An important issue to solve in future works concerns the formalization of the trade-off between robustness and performance. Open problems regard, also, how to design practical algorithms simple enough, robust and with good performance for industrial usage or how to design general ILC algorithm when having similar, but not identical, reference signals.

ILC field of activity is still growing and there are many theoretical and practical application issues to investigate.

1.1.3 ILC vs other control approaches

ILC differs from other learning control strategies which are described in the following.

Traditional Control

A *feedback* control strategy enables the system to be controlled to adjust its performance to meet a desired output, even when disturbances and uncertainties in the system model are present. Conventional feedback controllers, in fact, guarantee good tracking performance by reducing the sensitivity to disturbances. This is done by routing back, as inputs, the outputs of the system, as part of a chain of cause-and-effect that forms a loop. Since feedback controllers react to inputs and disturbances, they always have a lag in transient tracking.

A *feedforward* control can eliminate this lag and performs well when the system model is very accurate and signals are known or measurable. But, unmodeled dynamics and disturbances limit the effectiveness of feedforward control.

ILC comes in handy when one desires a control strategy that is robust to system uncertainties, works well in rejecting repeating disturbances and does not require exogenous signals, such as references and disturbances, to be known or measured, as a feedback strategy does; and that, at the same time, it is able to anticipates and compensates in advance, as a feedfarward control, undesired repeating terms, *i.e.* disturbances and uncertainties that affect the 'real' system dynamics, but are not accounted in the nominal ideal model.

In this sense, ILC behaves as a feedforward control in the time-domain and as a feedback control in the iteration-domain. In fact, the ILC update signal is based on data of previous iterations (feedback term) and is introduced directly in the system to compensate for repeating disturbances (feedforward term).

Some of the advantages of ILC over traditional control algorithms are:

High tracking performance: ILC is employed in applications in which precision in tracking desired trajectories is key.

Monotonic convergence: if convergence conditions are met, the system converges to a desired trajectory monotonically, preventing unwanted overshoots of the system trajectory.

High robustness: unlike other control methods, ILC requires little knowledge of the system to yield good results. It can be designed without an accurate model and can handle well model uncertainties, noise and external disturbances.

Advanced filtering and signal processing: because ILC deals with data time-sequences already available in its entirety, it can alleviate error sensitivity by using noncausal filtering technique and advanced signal processing (for instance, a zero-phase filtering allows for high-frequency attenuation without introducing lag).

The main disadvantage of ILC is linked to its inability to deal with relevant non-repeating disturbances that happen during trials. In fact, noise and non-repeating disturbances degrade ILC performance.

Because of that, ILC is generally used in combination with a feedback control: the iterative control is responsible for rejecting repetitive disturbances, while the feedback control has the task of reducing non-repeating disturbances and noise.

Repetitive Control

Repetitive Control (RC) is closely related to ILC. As with ILC, Repetitive Control exploits the information acquired in the past iterations to learn the way in which the system behaves, in order to reject repetitive disturbances.

The main difference between the two is that, RC is intended for continuous operations, where the next iteration immediately follows the current one[1], whereas ILC is designed for discontinuous operations. In fact, in RC, the system does not return to the initial conditions it had at the beginning of the trial before the next trial starts, unlike ILC.

This crucial difference makes the stability analysis very different. In fact, in time domain, RC acts as a feedback control and ILC acts as a feedforward control.

Additionally, ILC, unlike RC, works with finite time scale trials and infinite time scale in the iteration domain.

In a way, ILC can be considered a special case of RC if at the end of each iteration, the system is stopped and reset at the initial condition.

Intelligent Control

ILC can be classified in the of group Intelligent Control strategies. Machine Learning, Artificial Neural Networks and other ‘intelligent’ control strategies, as well as ILC, all implement a type of learning.

ILC differs from the so called ‘intelligent’ controllers because it is based on a system-theoretic approach that does guarantee a fast convergence, unlike the other strategies that requires extensive data training.

Adaptive Control

ILC differs from Adaptive Control strategies. Unlike ILC, the Adaptive Control adapts to a controlled system with parameters which vary, or are initially uncertain. The control modifies the control parameters rather than acting on the control input signal. Moreover, adaptive control typically does not take advantage of information contained in repetitive signals.

[1]The typical example of a RC application is the control of an hard disk drive’s head, in which the iteration is defined by the full rotation of the disk and where the next iteration immediately follows the current one.

1.1.4 Applications

Originally, ILC has been focused on improving the performance of systems that execute a single, repeated operation over and over again in time (*time* periodic). ILC has also found application to systems in which repetitiveness is not always with respect to time, but also with respect to *state* and to *iteration* (trajectory-dependent).

ILC is exploited in many practical industrial systems deployed in assembly lines, where mass production is key. ILC has been successfully applied in manufacturing (computer numerical control, motion systems, injection-molding machines, extruders, rolling mills, induction motors, chain conveyor system, engine valves), robotics (industrial robots, autonomous vehicles, braking systems), and chemical processing (thermal processing, chemical reactors).

It can also serve as a training mechanism for open-loop control and as part of an identification procedure (*e.g.* ILC is used in the experiments to obtain the aerodynamic drag coefficients of bullets).

In the aerospace field, practical ILC application are still under development.

Several works address the problem of augmenting the performance of UAVs in handling aggressive maneuvers by anticipating recurring disturbances (including the error in the model) and proactively compensating for them. Iterative learning control strategies can, in fact, be used to enable UAVs to learn from periodic maneuver's executions, to accurately track high-performance trajectories.

Lastly, ILC can be deployed in contexts in which repetitiveness is not guaranteed, such as, for example, in autonomous robots that perform non-repeating motions. For this application, for instance, ILC can be designed and trained with a number of repeating standard trajectories. The converged input that results in the 'best' performance is stored in a database for each different maneuver. When a new trajectory, that differs from one of the several stored in the database (only in space-scale and/or time-scale), is desired, the converged signal, previously stored in the database, is retrieved and used to shape the new signal. Practical applications in the robots with different task-dependent speeds is developed in several works.

In this thesis, an iterative learning control method for precise trajectory tracking of a quadrotor is developed.

1.2 General description of ILC

The typical Iterative Learning Control works as follows: at each iteration, the input update that is generated by the controller in the previous trial and that is stored in memory, enters the system and produces the output. The output signal is stored in memory. When the trial is over[2], ILC processes the data off-line and

[2]It is not always necessary to wait the end of the trial to process the data

updates the input. This new input is shaped so that it reduces the observed error, *i.e.* the difference between the actual output and the desired one. The updated input that exits the ILC is stored and applied to the system for the new iteration. The learning process continues with new trials.

A graphical description of the general ILC is displayed in Figure 1.1 [3].

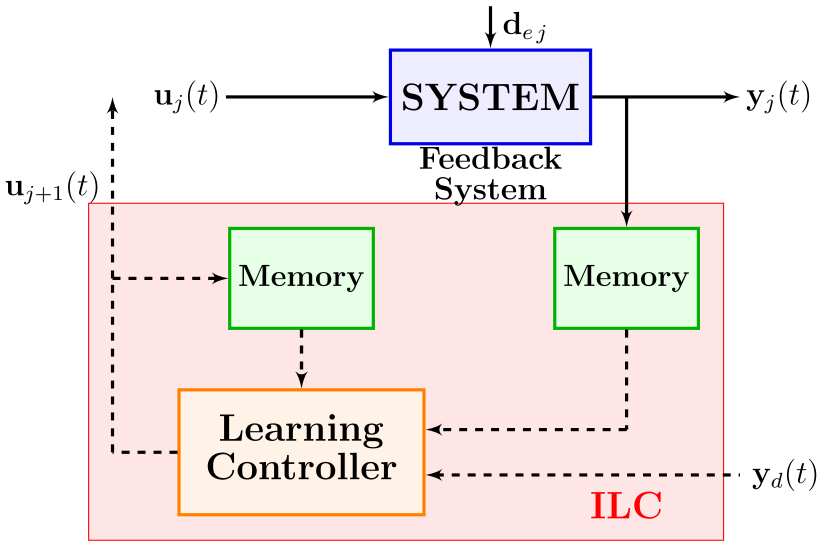


Figure 1.1: ILC general scheme

All signals are defined in the interval $t \in \mathcal{T}$, $\mathcal{T} := [0, T]$: the signal $\mathbf{u}_j(t)$ represents the input vector that enter the system, $\mathbf{d}_{e\,j}(t)$ represents external disturbances that perturb the system, while $\mathbf{y}_j(t)$ and $\mathbf{y}_d(t)$ represent respectively the signals of the actual and desired output. Subscript j indicates the trial number: $j \in \mathcal{J}$, $\mathcal{J} := \{0, 1, \ldots, N_j\}$.

The general ILC problem is to find a recursive law for the input update. This can be defined formally by introducing a mapping between the vector space of the input signals U and the one of the output signals Y:

$$f_S : U \mapsto V, \tag{1.1}$$

where f_S is a nonlinear operator defining the input-output relation of the given stable system S and it represents the system dynamics:

$$\mathbf{y}_j(t) = f_S\big(\mathbf{u}_j(t), t\big), \tag{1.2}$$

and with $\mathbf{u}_j(t) \in U$ and $\mathbf{y}_j(t) \in Y$.

The goal of the control is to drive the output to the desired value or equivalently to find the input $\mathbf{u}^*(t)$ that satisfied the minimization problem:

$$\mathbf{u}^*(t) = \underset{\mathbf{u}_j(t)}{\operatorname{argmin}} \|\mathbf{e}_j(t)\big)\| = \underset{\mathbf{u}_j(t)}{\operatorname{argmin}} \|\mathbf{y}_d(t) - f_S\big(\mathbf{u}_j(t), t\big)\|, \tag{1.3}$$

[3]The dashed lines in the figure are used to stress the fact that the signals are processed off-line.

where $\|\cdot\|$ is a suitable norm and and $\mathbf{e}_j(t)$ is the tracking error, *i.e.* the difference between the desired and actual output: $\mathbf{e}_j(t) = \mathbf{y}_d(t) - \mathbf{y}_j(t)$.

In order to find the input $\mathbf{u}_\infty(t)$, the ILC algorithm generates the input update with the following recursive law:

$$\mathbf{u}_{j+1}(t) = f\big(\mathbf{u}_{j'}(t'), \mathbf{y}_{j'}(t'), \mathbf{y}_d(t'), t\big), \tag{1.4}$$

with $t' \in \mathcal{T}$ and $j' \in \mathcal{J}' := \{j, j-1, \dots, 0\}$, so that:

$$\lim_{j\to\infty} \mathbf{u}_j(t) = \mathbf{u}_\infty(t), \tag{1.5}$$

or equivalently[4]:

$$\lim_{j\to\infty} \|\mathbf{e}_j(t)\| = 0. \tag{1.6}$$

In the most general description, the input update is a function of the time instant, the desired response and present and past inputs and outputs. Ideally, the input update of ILC algorithm should not depend on the desired trajectory $\mathbf{y}_d(t)$. In other terms:

$$\mathbf{u}_{j+1}(t) = f\big(\mathbf{u}_{j'}(t'), \mathbf{e}_{j'}(t'), t\big). \tag{1.7}$$

The value of the next iteration input at instant t, $\mathbf{u}_{j+1}(t)$, is therefore dependent on the time instant itself, and in the input and tracking error vectors; both are functions of the time-variable t', defined in the time-frame $[\,0, T\,]$ since there are no causality restrictions, and of the variable j', which specifies present and past iterations.

A number of postulate that underlie ILC approaches have been formulated in Arimoto's works [11] and are listed here.

P1: The iterations are finite in time: the time-period $T \in (0, \infty)$.

P2: The desired reference is defined *a priori* over time: $t \in \mathcal{T}$.

P3: Repetition of the initial conditions is satisfied: the system returns to its initial state after each trial: $\mathbf{x}_j(0) = \mathbf{x}_0$, $\forall j \in \mathcal{J}$

P4: The dynamics of the system is invariant throughout the repeated iterations;

P5: The output at every iteration $\mathbf{y}_j(t)$ can be measured, and so tracking error signal, *i.e.* the difference between the desired and the actual output $\mathbf{e}_j(t)$, can be used to generate the next input $\mathbf{u}_{j+1}(t)$.

P6: The dynamics of the system is causal and invertible (and also stable), that is, for a given desired output with a piece-wise continuous derivative , there exists a unique input $\mathbf{u}_{nom}(t)$ that drives the system to produce the output $\mathbf{y}_d(t)$.

[4]This is true only for some ILC formulation.

Postulates 3, 4, and 5 can be relaxed. Specifically:

P3*: Mismatch in initial conditions at the beginning of each iterations has a bounded error $\beta_1 > 0$: $\|\mathbf{x}_j(0) - \mathbf{x}_0\| < \beta_1$;

P4*: The norm of the input disturbance $\mathbf{d}_{e\,j}(t)$ induced during the repeated trials is limited: $\|\mathbf{d}_{e\,j}(t)\| < \beta_2$;

P5*: The output $\mathbf{y}_j(t)$ can be measured with noise and this noise is limited: $\|\boldsymbol{\mu}_j(t)\| < \beta_3$.

Many contributions discuss how to tackle the problem when one or several of the postulates are not satisfied.

The discussion, so far, was focused on continuous signals, *i.e.* signals dependent on the continuous time variable t. The same framework (with minimal changes) holds by replacing variable t with k when dealing with discrete-time signals: where $k \in \mathcal{K} := \{0, 1, \ldots, N-1\}$, and $N < \infty$ is the finite trial length.

1.3 ILC algorithms

The ILC algorithm defines the input update control law of ILC. They can be distinguished by a number of properties such us the ones listed below.

Linearity *Linear* and *nonlinear* algorithms are distinguished.

ILC order Based on the order of the ILC algorithm, which defines the number of iterations used (in terms of measurements) to update the ILC input, *first-order* algorithms are separated from *higher-order* algorithms.

Causality *Causal* and *non-causal* control algorithms are defined.

System description Depending on the system description, the algorithms are categorized in *continuous*-time and *discrete*-time, as well as in *frequency*-domain and *time*-domain.

In this section, an overview of the main ILC algorithms is presented, with a particular focus on linear first-order algorithms applied to single-input, single-output (SISO) discrete-time linear systems. The ‘classical’ ILC formulation is then discussed. The description of the system to which the control is applied is postponed to the next section.

1.3.1 Linear and nonlinear algorithms

Nonlinear algorithms

As stated, ILC update laws can be distinguished in linear and nonlinear algorithms. A general nonlinear ILC algorithm is given by the equation:

$$u_{j+1}(t) = f\big(u_j(t), u_{j-1}(t) \ldots, u_1(t), u_0(t), e_j(t), e_{j-1}(t), \ldots, e_1(t), e_0(t)\big) \tag{1.8}$$

where, as always, u and e represents, respectively, input and tracking error (of a SISO system), j is the iteration index and t is the continuous time variable. This 'continuous' formulation is defined with respect to the continuous time variable t; it can be replaced by the discrete-time variable k for the discrete case.

In this algorithm, the input update $u_{j+1}(t)$ is a nonlinear function of previous iterations information, but, in general, it can also depend on the future iteration input or the estimated predicted tracking error. The class of nonlinear algorithms is very large and the formulations available are applied on the basis of the problem under study. In fact, the category of nonlinear ILC algorithms is still an open area of research.

Linear algorithms

Linear algorithms, *i.e.* algorithms in which the function f in (1.8) is linearly dependent in past inputs and error measurements, are more commonly applied both on linear and nonlinear system.

1.3.2 First-order and Higher-order ILC algorithms

First-order algorithms

A *first-order* ILC algorithm uses an update formula that only exploits measurements of the current iteration to generate the new input update:

$$u_{j+1}(t) = f\big(u_j(t), e_j(t)\big), \tag{1.9}$$

High-order ILC algorithms

A so called *high-order* ILC algorithm, instead, uses measurements from current and previous iterations to shape the ILC update:

$$u_{j+1}(t) = f\big(u_j(t), \ldots, u_{j-N_o+1}, e_j(t), \ldots, e_{j-N_o+1}(t)\big), \tag{1.10}$$

where N_o defines the order of the ILC that is $N_o \geq 2$.

1.3.3 Continuous and discrete algorithms

Continuous algorithms

Originally, ILC was formulated for continuous systems in a *continuous* time framework. The first ILC algorithm proposed by Arimoto, in fact, is in continuous time:

$$u_{j+1}(t) = u_j(t) + \Gamma \dot{e}_j(t), \tag{1.11}$$

where Γ is a constant gain that multiplies the derivative of the continuous tracking error in time.

The generalized algorithm is:

$$u_{j+1}(t) = u_j(t) + \Phi e_j(t) + \Psi \int_0^t e_j(\tau)\, d\tau + \Gamma \dot{e}_j(t), \tag{1.12}$$

where the constant gains Φ, Ψ and Γ represents respectively the proportional (P), integral (I) and derivative (D) terms of the error that update the input of the PID-like ILC.

Discrete algorithms

The *discrete*-time counterpart of the control law (1.11) is the noncausal[5] update:

$$u_{j+1}(k) = u_j(k) + \gamma e_j(k+1), \tag{1.13}$$

where γ is a constant gain (the gain can also be a function of the discrete time variable k) and where the error is defined at the next time instant.

Discrete time algorithms are preferred to the continuous because the discrete domain is the natural domain for ILC. In fact, input and output signals are sampled and stored in memory so that past-information data can be retrieved.

The algorithm (1.13) can be seen as a special case of a more general structure, by introducing a filter $L(q)$, known as the learning function, and a so called Q-filter $Q(q)$, where q is the forward time-shift operator.

The learning function maps the error signal $e_j(k+1)$ to the control signal, while the Q-filter filters the ILC updating signal $u_j(k)$ or, more often, the computed learning signal $u_j(k) + L(q)e_j(k)$ before reapplying it to the system.

The common formulation of linear discrete ILC algorithms is, thus:

$$u_{j+1}(k) = Q(q)\Big(u_j(k) + L(q)e_j(k+1)\Big), \tag{1.14}$$

The scheme is illustrated in Figure 1.2.

The formulation takes different forms depending on the system descriptions (time-domain or frequency-domain, SISO time- and iteration-domain description or MIMO iteration-domain description). These formulations are later explained.

[5]The difference in causal and noncausal ILC is later explained.

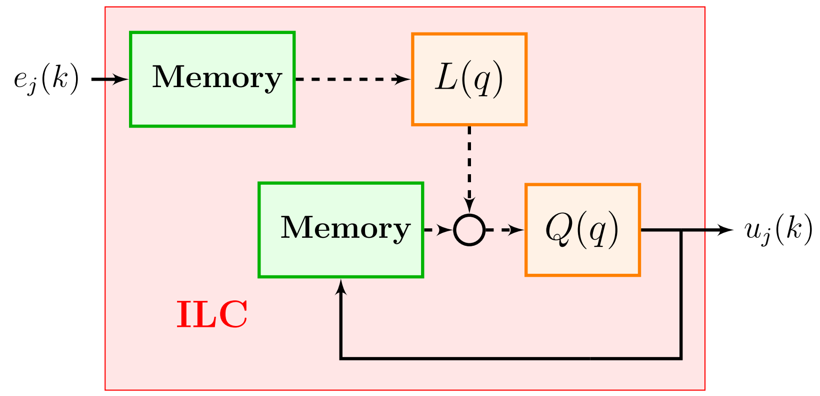


Figure 1.2: General ILC algorithm scheme

The general ILC algorithm (1.14) can be extended to include a feedback control term:

$$u_{j+1}(k) = Q(q)\Big(u_j(k) + L(q)e_j(k+1)\Big) + C(q)e_{j+1}(k) \tag{1.15}$$

This control algorithm is known as the *current-iteration* ILC. Redefining the Q-filter and the learning function it is equivalent to the general ILC algorithm 1.14 combined with a feedback control in parallel[6].

Causal and Noncausal Learning

ILC can anticipate and promptly respond to repeated disturbances. This characteristic depends on the causality of the learning algorithm.

Definition: An ILC learning algorithm is *causal* if the input update $\mathbf{u}_{j+1}(t)$ depends only on $\mathbf{u}_j(t')$ and $\mathbf{e}_j(t')$ for $t' \leq t$. It is *noncausal* if $\mathbf{u}_{j+1}(t)$ is also a function of $\mathbf{u}_j(t')$ or $\mathbf{e}_j(t')$ for some $t' > t$.[7]

Because of the recursive nature of ILC, the entire time sequence of data of all past iterations is available. This allows the implementation of a *noncausal* control learning strategy.

The noncausal algorithm has the advantage to anticipate repeating disturbances and promptly compensates them with a proper input update, by knowing in advance, from data available in previous iterations, where the system is going to. For this reason, the noncausal update law:

$$u_{j+1}(k) = u_j(k) + Le_j(k+1), \tag{1.16}$$

is preferred to the causal counterpart:

$$u_{j+1}(k) = u_j(k) + Le_j(k). \tag{1.17}$$

[6]Feedback control used with ILC is explained in subsection 1.6.1

[7]This definition is referred to the continuous signals in time, the same holds for the discrete signals by replacing t with k and t' with k'.

In fact, the input update in equation (1.17) at time instant k aims to reduce the error $\mathbf{e}_j(k)$, which depend on the disturbance $\mathbf{d}_j(k)$ (disturbance at the iteration j at instant k), by acting on the same time instant; whereas, the noncausal update law in equation (1.16), uses information of the current iteration error at the next time step $\mathbf{e}_j(k+1)$, to reduce the disturbance $\mathbf{d}_j(k+1)$, by shaping the next iteration input at the current time step $\mathbf{u}_{j+1}(k)$.

In this sense, in a noise-free scenario, there is an equivalence between feedback control and causal ILC, since, like a feedback control, it simply reacts to errors at the same time instant. No equivalent feedback controller can, instead, provide the same control action as the converged noncausal ILC.

1.4 System description

In this section, first, a brief description of the system to be controlled is given, then, starting from the general ILC algorithm, the analysis is distinguished based on the system representation. In the next chapter, a more detailed presentation focusing on the specific system studied is reported.

In order to show convergence and to ensure stability in actual implementations, the system is required to be stable, as specified in the principles underlying the concept of ILC postulated by Arimoto. The following analysis is centred in LTI systems. If the plant is initially unstable, it should first be stabilized by a conventional feedback control technique. This agrees with the focus of the ILC algorithm that is to improve the performance of the system.

The system considered is represented in the discrete-time-domain, that is the ILC 'natural' domain, as previously mentioned. Specifically, it is considered a strictly proper Linear Time Invariant (LTI) Single Input, Single Output (SISO) discrete-time system. This nominal model approximating the 'actual' system[8], is written in the state-space form:

$$\begin{cases} \tilde{\mathbf{x}}_j(k+1) = \mathbf{A}\tilde{\mathbf{x}}_j(k) + \mathbf{B}u_j(k) \\ \tilde{y}_j(k) = \mathbf{C}\tilde{x}_j(k), \end{cases} \tag{1.18}$$

where:

k is the discrete time index;

j is the trial number;

$u_j(k)$ is the input control signal;

$y_j(k)$ is the actual output control signal;

[8]The tilde is used to distinguish the nominal model from the 'actual' model.

Given the initial state condition $\tilde{\mathbf{x}}_j(0) \approx \mathbf{x}_0, \forall j$, the nominal system (1.18) can be rewritten as follows:

$$\tilde{y}_j(k) = \underbrace{\mathbf{C}(q\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}}_{\tilde{P}(q)}\, u_j(k) + \underbrace{\mathbf{C}\mathbf{A}^k}_{d^0(k)}\mathbf{x}_0, \tag{1.19}$$

where:

$d_j(k)$ is the disturbance that includes external perturbations and model uncertainties and noise measurements. It can be approximated as an exogenous signal that repeats at each iteration if noise and non-repeating error model are neglected: $d_j(k) \approx d^0(k) + d^{R-}(k)$, where $d^0(k)$ is the disturbance of the free response of the system due to non-null initial conditions, and $d^{R-}(k)$ is remaining repeatable term of the disturbance;

q is the forward time-shift operator: $qx(k) = x(k+1)$;

$\tilde{P}(q)$ is a proper rational function of q that represent the *nominal* Asymptotically Stable (AS) system and has a delay, or equivalently relative degree, of m;

$d^0(k)$ is the disturbance of the free response of the system due to non-null initial conditions.

The 'real' model is instead:

$$y_j(k) = \tilde{P}(q)u_j(k) + d_j(k), \tag{1.20}$$

where $d_j(k)$ is the disturbance that includes external perturbations and model uncertainties and noise measurements. It can be approximated as an exogenous signal that repeats at each iteration if noise and non-repeating error model are neglected: $d_j(k) = d_j^{NR}(k) + d_j^R(k) \approx d_j^R(k) = d^0(k) + d^{R-}(k)$, where $d^0(k)$ is the disturbance of the free response of the system due to non-null initial conditions, and $d^{R-}(k)$ is remaining repeatable disturbance;

Considering a sequence of N sampling instants, the input, output, disturbance and desired reference signals are so defined:

$$\begin{aligned} &u_j(k), \quad k \in \{0, 1, \ldots, N-1\} \\ &y_j(k), \quad k \in \{m, m+1, \ldots, N+m-1\} \\ &d_j(k), \quad k \in \{m, m+1, \ldots, N+m-1\} \\ &y_d(k), \quad k \in \{m, m+1, \ldots, N+m-1\}. \end{aligned}$$

N is always a finite integer, since the trials are limited in time. However, it can be considered infinite ($N = \infty$) for analysis purposes.

Starting from the basic system description, the ILC analysis, based on the general ILC learning update algorithm (1.14), differentiates into two separate path, depending on the representation used.

1.4.1 Time-domain analysis: lifted-system representation

When the system is described in the *time*-domain, the so called *lifted*-representation is preferred in describing the input/output relation and the ILC update algorithm. The lifted form, in fact, allows to write the SISO time and iteration-domain dynamic system as a multiple-input, multiple-output (MIMO) iteration-domain dynamic system.

To construct the lifted-system, the nominal plant $\tilde{P}(q)$ is, first, expanded as an infinite power series[9], by dividing its denominator into its numerator:

$$\tilde{P}(q) = \tilde{p}_1 q^{-1} + \tilde{p}_2 q^{-2} + \dots, \tag{1.21}$$

where the coefficients $\tilde{p}_k$ are the Markov parameters and altogether define the impulse response of the system; these coefficient are equal to $\tilde{p}_k = \mathbf{C}\mathbf{A}^{k-1}\mathbf{B}$, assuming $\tilde{p}_1 \neq 0$ and the relative degree of the plant equal to $m = 1$.

Then, the input/output dynamics in (1.20) can be rewritten as:

$$\mathbf{y}_j = \tilde{\mathbf{P}}\mathbf{u}_j + \mathbf{d}_j. \tag{1.22}$$

The vectors $\mathbf{y}_j$, $\mathbf{u}_j$ and $\mathbf{d}_j$ are shifted by one time step to accommodate the one-step delay in the plant ($m = 1$) and are so defined:

$$\begin{aligned}
\mathbf{y}_j &= \left[\, y_j(1), y_j(2), \dots, y_j(N) \,\right]^T, \\
\mathbf{u}_j &= \left[\, u_j(1), u_j(2), \dots, u_j(N) \,\right]^T, \\
\mathbf{d}_j &= \left[\, d_j(1), d_j(2), \dots, d_j(N) \,\right]^T,
\end{aligned}$$

while lower-triangular Toeplitz matrix $\tilde{\mathbf{P}}$ is:

$$\tilde{\mathbf{P}} = \begin{bmatrix} \tilde{p}_1 & 0 & \cdots & 0 \\ \tilde{p}_2 & \tilde{p}_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ \tilde{p}_N & \tilde{p}_{N-1} & \cdots & \tilde{p}_1 \\ . & & & \end{bmatrix}$$

Also the tracking error is redefined:

$$\mathbf{e}_j = \mathbf{y}_d - \mathbf{y}_j, \tag{1.23}$$

with $\mathbf{y}_d = \left[\, y_d(1), y_d(2), \dots, y_d(N) \,\right]^T$.

Analogously, by defining the non-causal Q-filter and learning function as two finite impulse response (FIR) filters (whose impulse response length depends on the trial length N):

$$Q(q) = \cdots + q_{-2}\mathrm{q}^{-2} + q_{-1}\mathrm{q}^{-1} + q_0 + q_1\mathrm{q} + q_2\mathrm{q}^2 \cdots, \tag{1.24}$$

[9]In the next chapter the lifted-form is derived directly from the state space representation.

and

$$L(q) = \cdots + l_{-2}\mathrm{q}^{-2} + l_{-1}\mathrm{q}^{-1} + l_0 + l_1\mathrm{q} + l_2\mathrm{q}^2 \cdots , \tag{1.25}$$

the general ILC algorithm (1.14) takes the form of:

$$\mathbf{u}_{j+1} = \mathbf{Q}\Big(\mathbf{u}_j + \mathbf{L}\mathbf{e}_j\Big), \tag{1.26}$$

where the $\mathbf{Q}$ and $\mathbf{L}$ matrices are:

$$\mathbf{Q} = \begin{bmatrix} q_0 & q_{-1} & \cdots & q_{-(N-1)} \\ q_1 & p_0 & \cdots & q_{-(N-2)} \\ \vdots & \vdots & \ddots & q_{-1} \\ q_{N-1} & p_{N-2} & \cdots & p_0 \end{bmatrix},$$

and

$$\mathbf{L} = \begin{bmatrix} l_0 & l_{-1} & \cdots & l_{-(N-1)} \\ l_1 & l_0 & \cdots & l_{-(N-2)} \\ \vdots & \vdots & \ddots & l_{-1} \\ l_{N-1} & l_{N-2} & \cdots & l_0 \end{bmatrix}.$$

This system description is very general and it can handle also Linear Time-Varying (LTV) systems.

1.4.2 Frequency-domain analysis: the z-domain representation

Introducing the *z-transform* of a discrete-time signal, the system dynamics in equation (1.20) can be represented in the z-domain:

$$Y_j(z) = \hat{P}(z)U_j(z) + D_j(z). \tag{1.27}$$

where z is a complex variable and the capital letters represent the signals in the z-domain: $U_j(z) = \sum_{k=0}^{\infty} u_j(k)z^{-k}$, $Y_j(z) = \sum_{k=0}^{\infty} y_j(k)z^{-k}$, $D_j(z) = \sum_{k=0}^{\infty} d_j(k)z^{-k}$, with the signals $u_j(k)$, $y_j(k)$ and $d_j(k)$ defined for $k \in [\,0, \infty\,)$. Actually, the signals are finite as the trials have a limited duration ($N_j < \infty$); therefore, when representing the signal in z-domain, an approximation is done. The frequency response of the system is obtained by replacing z with $e^{i\theta}$ for $\theta \in [\,-\pi, \pi\,]$.

With this system description, the general ILC input update algorithm in (1.14) is formulated as following:

$$U_{j+1}(k) = Q(z)\Big(U_j(z) + zL(z)E_j(z)\Big), \tag{1.28}$$

where $E_j(z) = Y_d(z) - Y_j(z)$, with $Y_d(z) = \sum_{k=0}^{\infty} y_d(k)z^{-k}$.

1.5 Analysis of performance

In this section, an analysis of the general discrete linear ILC scheme is presented.

Recalling the general algorithm (1.14), the objective of the designer is to choose the Q-filter and learning function that regulate how the iterative learning control performs. Q-filter and learning function, independently from the representation, can be constant- or iteration-varying.

The learning function, as the name suggests, represents the learning contribution of the algorithm, that is the unfiltered ILC input update term.

The role of the Q-filter is, instead, to limit the frequency range of the learning for ILC stability and noise attenuation. In fact, Q-filtering involves a trade-off between minimizing repeatable error and amplifying noise. When $Q(q) = 1$ (or $Q(z) = 1$ in the frequency-domain or $\mathbf{Q} = \mathbf{I}$ in matrix form), the repeatable error is eliminated, but noise and non-repeatable disturbances are amplified. Vice versa, when $Q(q)$ (or $Q(z)$ or $\mathbf{Q}$) is null, it has no effect on rejecting disturbances, both repeatable and non-repeatable (noise). Therefore, the choice of the Q-filter should be based on the nature of the disturbances: should be set to one in frequency regions where the repeatable component of the disturbance is dominant, and set to zero where non-repeatable component is relevant.

The typical Q-filter design is a low pass filter. This is because the (feedback) system model is generally well known at low frequencies, where repeating disturbances are dominant, and uncertain at higher frequencies, where noise is relevant and where the feedback control is predominant over ILC.

The characteristics of the filter, such as the type, the order or the cut-off frequency, are selected by the designer to achieve (robust) stability and high tracking (robust) performance. Additional limitations can be introduced to include also input constraints.

To make it clear, ILC stability and performance definitions are briefly presented.

Stability (convergence)

The asymptotic stability (AS) of the system determines the effectiveness of the iterative control. The ILC system is AS if $\exists\, \bar{u} \in \mathbb{R} : |u_j(k)| \leq \bar{u},\ \forall k \in \mathcal{K}, \forall j \in \mathcal{J}$, and:

$$\exists \lim_{j\to\infty} u_j(k), \quad \forall k \in \mathcal{K}, \tag{1.29}$$

where $\mathcal{K} := \{\, 0, \dots, N-1 \,\}$ and $\mathcal{J} := \{\, 0, \dots, N_j \,\}$; where the converged control input is:

$$u_\infty(k) := \lim_{j\to\infty} u_j(k). \tag{1.30}$$

Based on the system description and ILC algorithm, different stability analysis are considered. If the *lifted-form* ILC formulation (1.26) is taken into account, the system dynamics and ILC update can be rewritten as:

$$\mathbf{u}_{j+1} = \mathbf{Q}(\mathbf{I} - \mathbf{L}\tilde{\mathbf{P}})\mathbf{u}_j + \mathbf{Q}\mathbf{L}(\mathbf{y}_d - \mathbf{d}), \tag{1.31}$$

by placing the system dynamics (1.20) in the error definition (1.23) and by substituting this error in (1.26) [10].

Thus, the AS, *i.e.* the *convergence*, of the ILC system is linked to eigenvalues of the matrix $\mathbf{Q}(\mathbf{I} - \mathbf{L}\mathbf{P})$. The ILC system is stable if and only if:

$$\rho\big(\mathbf{Q}(\mathbf{I} - \mathbf{L}\tilde{\mathbf{P}})\big) < 1 \tag{1.32}$$

where ρ stands for the spectral radius. If the maximum singular value of the matrix is lower than one, the condition gives monotone convergence. The convergence speed, is thus related to the maximum singular value of the matrix, the smaller it is the higher the speed.

Analogously, the convergence of the ILC system equations (1.28) and (1.27), in the frequency-domain, results in the following condition:

$$\|Q(z)(1 - zL(z)\tilde{P}(z))\|_\infty < 1 \tag{1.33}$$

where the $\mathcal{H}_\infty$-norm of a matrix $M(z)$ is defined as: $\|M(z)\|_\infty = \sup_{\omega\in[-\pi,\pi]} |M(e^{i\omega})|$. The value of the $\mathcal{H}_\infty$-norm is linked to the convergence speed: the smaller it is, the faster is the convergence.

Considering the general ILC algorithm (1.14), the convergence is satisfied if a proper choice of Q-filter and learning function are considered.

Performance

The performance of an ILC system is based on the asymptotic value of the error. If the system is AS, the converged error exists and it is defined as:

$$e_\infty(k) := \lim_{j\to\infty} e_j(k). \tag{1.34}$$

As commented before, the performance analysis distinguishes based on the ILC system description. For the lifted-domain representation, the asymptotic error is computed from the converged input update, obtained by solving equation (1.31) for $j \to \infty$:

$$\mathbf{e}_\infty = \big[\mathbf{I} - \tilde{\mathbf{P}}[\mathbf{I} - \mathbf{Q}(\mathbf{I} - \mathbf{L}\tilde{\mathbf{P}})]^{-1}\mathbf{Q}\mathbf{L}\big](\mathbf{y}_d - \mathbf{d}). \tag{1.35}$$

For the z-domain ILC system, the asymptotic error is, instead:

$$E_\infty(z) = \frac{1 - Q(z)}{1 - Q(z)[1 - zL(z)\tilde{P}(z)]}[Y_d(z) - D(z)]. \tag{1.36}$$

[10] The disturbance $\mathbf{d}$ has been considered iteration-independent, *i.e.* noise is neglected.

In this last formulation, the error performance can be expressed, for example, through a performance weight $W_p(z)$ as:

$$\left| W_p(z) \frac{1 - Q(z)}{1 - Q(z)[1 - zL(z)\tilde{P}(z)]} \right| < 1, \, \forall z \in \mathcal{D}, \tag{1.37}$$

where: $\mathcal{D} := \{ z = e^{j\omega} \big| \quad \omega \in [-\pi, \pi] \}$.

When recurring to 'non-standard' ILC formulations, the performance is more commonly evaluated comparing the initial trial error $e_0(k)$ with the asymptotic error $e_\infty(k)$, using either qualitative or quantitative methods (e.g. Root Mean Square).

The stability and performance analysis can be repeated to include *robustness* analysis. Analogous definitions are derived. In those definitions, the term $\tilde{\mathbf{P}}$ used to represent the nominal model is replaced with $\mathbf{P}_\Delta$, to indicate a set of (stable)-uncertain ILC systems, and the constraints are referred to the worst-case condition.

1.6 Typical design methods

In this section the typical ILC design techniques implemented in real systems are discussed and analysed: first a subsection is dedicated to illustrate how to use feedback control in combination with ILC, then a distinction between basic and model-based design methods is discussed and presented in two separate subsections: for both designs some examples are given, with the discussion limited to discrete ILC algorithms only.

1.6.1 Feedback control with ILC

As explained numerous times, the goal of ILC is to track a desired reference and reject repeating disturbances. In other words, the objective of the iterative control is to generate an open-loop input signal that approximately inverts the system's dynamics and ideally removes only the repeating disturbance term, ignoring noise and non-repeating contributions.

Since it has no feedback mechanism to respond to unexpected non-repeating disturbances, in many physical implementations, a well-designed feedback controller is used in combination with ILC.

Most systems, in fact, feedback controllers are already present in the system in order to stabilize the plant in question and reduce the sensitivity to disturbances. Two are the possible installation arrangements:

Serial arrangement When ILC is implemented in a pre-existing system, in which direct access to the control signal that feeds the plant is inconvenient or forbidden, as for many commercial controllers, it is generally installed in a so called *serial* arrangement, see Figure 1.3. In this case, ILC control input is added to the reference before the feedback loop.

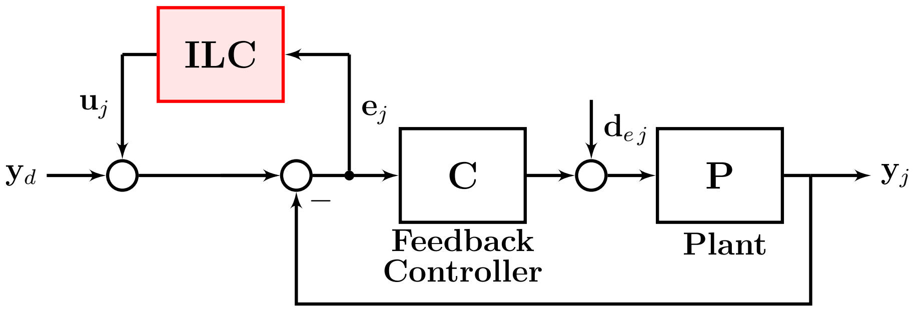


Figure 1.3: ILC in *serial* arrangement

Paralel arrangement If, instead, the ILC control input is combined with the feedback control input, as a feedforward signal, and applied directly to the plant, the arrangement is defined as *parallel.* This approach, illustrated in Figure 1.4, is more intuitive given that ILC acts directly on the control variables.

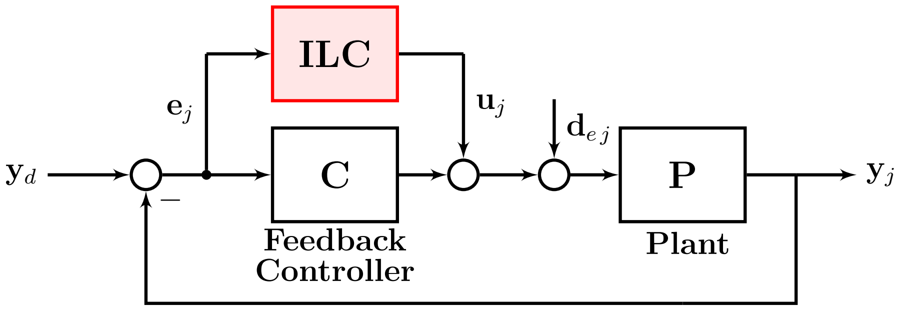


Figure 1.4: ILC in *parallel* arrangement

It is worth pointing out that, whenever a feedback control is already embedded in the system, an iterative learning controller can be installed without altering the feedback control parameters. In fact, by adding the ILC, independently from the chosen configuration, the output of the system, dependent on the feedforward input update, is summed to the output resulting from the feedback loop. Setting the ILC input to zero, *i.e.* disconnecting the ILC, in both cases results in the standard feedback-controlled response to the desired reference. Therefore, in both arrangements, ILC can be disabled whenever non-repeating reference trajectories are used.

1.6.2 Basic design methods

Basic ILC design techniques, differently from model-based methods, are very simple to implement as they require minimal knowledge of the system studied and are not computationally demanding. Because of that, these methods are appealing from the industrial perspective where simplicity is key.

Design methods of this kind, relying on very little information of the system, are generally called *model-free* or *heuristic* ILC.

The pros of basic ILC approaches, as mentioned earlier, is the pure simplicity of the algorithms: they depend on few parameters. This is motivated by the practical implementation, as well as by the desire to control a system without knowing its entire dynamics. On the cons, these algorithms do not perform as well as model-based algorithms.

Examples of basic design methods are based, for instance, on the Arimoto-type algorithms or in model-free self-tuning algorithms.

Heuristic design

A typical heuristic design method is presented. As an example, consider a SISO system that needs to be controlled where only time delay and the static gain are known.

The discrete ILC update law is:

$$u_{j+1}(k) = Q(q)\left[u_j(k) + \gamma q^{\delta} e_j(k)\right], \tag{1.38}$$

where the Q-filter determines how large is the part of the repeating error dynamics that should be learned, while the learning function $L(q) = \gamma q^{\delta}$ compensates for time delay through a forward time shift operator q weighted for a constant gain γ.

These terms are computed following the simple procedure in Table 1.1.

	Design procedure
Step 1.	Choose the filter $Q(q)$ as a low-pass filter with cutoff frequency such that the bandwidth of the learning algorithm is sufficient.
Step 2.	Let $L(q) = \gamma q^{\delta}$ be the learning function. Select the learning gain γ and time shift δ such that the frequency-domain stability criterion is satisfied.

Table 1.1: An heuristic design algorithm

1.6.3 Model-based design methods

Model-based design methods require an explicit model of the system [12]. Because of that, the algorithms they are based on, tend to be more computationally intensive, with the benefit of improving the overall system performance.

Plant inversion design methods

Plant inversion methods use models of the inverted plant (or system) dynamics as the learning function $L(q)$.

An example of a discrete-time plant inversion ILC algorithm is the following:

$$u_{j+1}(k) = u_{j+1}(k) + \tilde{P}^{-1}(q)e_j(k) \tag{1.39}$$

where $\tilde{P}(q)$ indicates the nominal model of the actual plant $P(q)$.

In these methods, convergence occurs in a single iteration and the converged error e_∞ is null.

The problem of plant inversion design methods is linked to the practical difficulties in implementing such algorithms. In fact, the discrete-time equivalent of a continuous system is very often non-minimum phase. Therefore, direct inversion of the non-minimum phase plant results in an unstable filter (where undesirably large control signal variations can occur). Moreover, these methods depend consistently on the accuracy of the model. Mismatch between nominal and actual model, that are usually consistent at higher frequencies, can lead to poor transient behaviour and may violate the stability properties of the system.

Frequency-domain design methods

A branch of model-based design methods is based on frequency-domain analysis algorithms [13].

Typical examples are $\mathcal{H}_\infty$ methods.

$\mathcal{H}_\infty$-design methods use a systematic approach to ILC design. Given the classical ILC update law in equation (1.14), the design goal is to find the best learning function $L(q)$ for a given filter $Q(q)$. In other terms, these algorithms uses $\mathcal{H}_\infty$ synthesis to minimize the error transmission from one iteration to the next by finding the learning function that gives the fastest convergence rate for the given Q-filter choice. In mathematical terms, $L^*(q)$ is the solution of the synthesis problem that results in the minimum convergence speed γ^* (that has to be below unity):

$$\|Q(z)\big(I - zL^*(z)\tilde{P}(z)\big)\|_\infty = \gamma^* < 1 \tag{1.40}$$

The algorithm is summarized in Table 1.2.

The model matching problem can be written equivalently as a lower Linear Fractional Transform (LTF) that defines the transfer function of the error transmission from one iteration $E_j(z) - E_\infty(z)$ to the next $E_{j+1}(z) - E_\infty(z)$:

$$\mathcal{F}_l(\mathbf{G}, L) = G_{11}(z) + G_{12}(z)L(z)\big(I - G_{22}(z)L(z)\big) \tag{1.41}$$

where $G(z)$ is equal to:

$$\mathbf{G}(z) = \begin{bmatrix} G_{11}(z) & G_{12}(z) \\ G_{21}(z) & G_{22}(z) \end{bmatrix} = \begin{bmatrix} Q(z) & Q(z) \\ -\tilde{P}(z) & 0 \end{bmatrix}.$$

	Design procedure
Step 1.	Choose the filter $Q(q)$ to be a low-pass weighting filter with pre-specified cut-off frequency ω_c, such that $Q(e^{i\omega}) = I$, $\forall\omega \in [0, \omega_c]$, and $Q(e^{i\omega}) = 0$, $\forall\omega > \omega_c$.
Step 2.	For given $\tilde{P}(z)$ and chosen $Q(z)$, let $L(z)$ be the solution of the (sub)optimal $\mathcal{H}_\infty$ synthesis problem: $L^*(z) = \underset{L\in\mathcal{H}_\infty}{\arg\min} \lVert Q(z)\big(I - zL(z)\tilde{P}(z)\big)\rVert_\infty$

Table 1.2: An $\mathcal{H}_\infty$ design algorithm

The scheme is illustrated in Figure 1.5 .

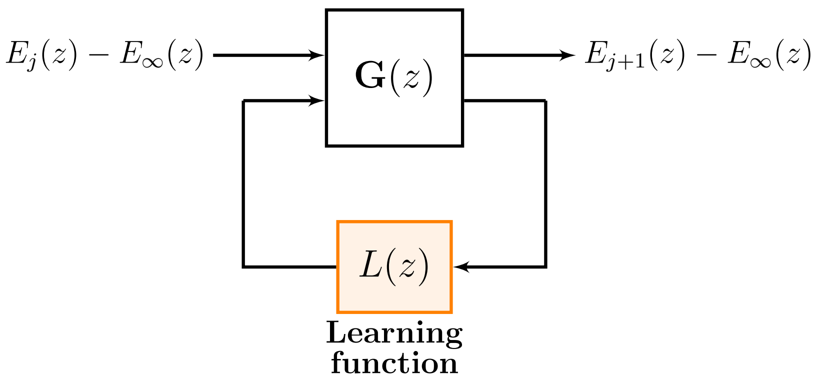


Figure 1.5: $\mathcal{H}_\infty$- synthesis scheme

Alternative $\mathcal{H}_\infty$-design methods include also robust performance analysis by incorporating known uncertainty bounds. The solutions are however limited to causal functions only.

Optimization-based design methods

Optimization-based design methods, also known as *norm-optimal* approaches, exploit the numerical optimization context to update the ILC signal for the next iteration. These algorithms use the lifted-form description of the system and the matrix representation of the ILC update algorithm.

The aim of the optimization correction is to reduce at each iteration step the future tracking error while avoiding high actuator demands, *i.e.* by penalising the change of the ILC input signal between successive iterations and the new input signal itself.

The ILC input signal $\mathbf{u}_{j+1}$ is so the solution of a minimization problem. It is derived from the minimization of a cost function, generally based on the predicted next iteration error (which in turn depend on the updated input and the current cycle error), the new ILC input signals, the magnitude of the input signal itself and the optimization weights:

$$J_{j+1}(\mathbf{u}_{j+1}) = f_c\big(\mathbf{e}_{j+1}(\mathbf{u}_{j+1}, \mathbf{e}_j), \mathbf{u}_{j+1}, \mathbf{u}_j, weights\big), \tag{1.42}$$

where the *weights* are positive-(semi)definite matrices that can be both constant or iteration dependent.

Chosen the weights, the current input data $\mathbf{u}_j$ and the current error measurement vector $\mathbf{e}_j$, the cost function at the next iteration is ultimately a function of $\mathbf{u}_{j+1}$. So, minimizing the cost criterion means finding the updated input vector.

Typical examples of these methods are the Quadratically Iterative Learning Control (Q-ILC) and the estimation-based norm-optimal ILC methods, such as the Kalman filter enhanced-ILC object of the thesis.

To explicitly consider multiple ILC objectives (*i.e.* convergence speed, robust convergence, converged tracking performance and input constraint), the so called *multi-objective* methods were recently developed. The theory behind these methods is later presented.

Q-ILC design methods In Q-ILC algorithms, the optimization problem consists in minimizing a quadratic next iteration cost function as the following:

$$J_{j+1}(\mathbf{u}_{j+1}) = \|\mathbf{e}_{j+1}\|_{\mathbf{W}_{e\,j}} + \|\mathbf{u}_{j+1} - \mathbf{u}_j\|_{\mathbf{W}_{\Delta u\,j}} + \|\mathbf{u}_{j+1}\|_{\mathbf{W}_{u\,j}}. \tag{1.43}$$

where the norms are 'weighted'. Written more explicitly it becomes:

$$\begin{aligned} J_{j+1}(\mathbf{u}_{j+1}) = \mathbf{e}_{j+1}^T \mathbf{W}_{e\,j} \mathbf{e}_{j+1} + (\mathbf{u}_{j+1} - \mathbf{u}_j)^T \mathbf{W}_{\Delta u\,j} (\mathbf{u}_{j+1} - \mathbf{u}_j) \\ + \mathbf{u}_{j+1}^T \mathbf{W}_{u\,j} \mathbf{u}_{j+1}. \end{aligned} \tag{1.44}$$

The estimated tracking error of the next iteration can be defined as

$$\hat{\mathbf{e}}_{j+1} = \mathbf{e}_j - \tilde{\mathbf{P}}(\mathbf{u}_{j+1} - \mathbf{u}_j) \tag{1.45}$$

where the hat symbol on the error indicates that the estimated error is based on the nominal model $\tilde{\mathbf{P}}$ only and does not take into model uncertainties and undesired disturbances: $\hat{\mathbf{e}}_j = \mathbf{y}_d - \mathbf{y}_j = \mathbf{y}_d - \tilde{\mathbf{P}}\mathbf{u}_j$.

Solving the optimization problem for the next iteration estimated error considered, that is, deriving the cost function with respect to the $\mathbf{u}_{j+1}$ and imposing the point to be stationary, the ILC algorithm becomes:

$$\mathbf{u}_{j+1} = \mathbf{Q}_{opt}\mathbf{u}_j + \mathbf{L}_{opt}\mathbf{e}_j, \tag{1.46}$$

where the optimal Q-filter is:

$$\mathbf{Q}_{opt} = \big(\tilde{\mathbf{P}}^T \mathbf{W}_{e\,j} \tilde{\mathbf{P}} + \mathbf{W}_{\Delta u\,j} + \mathbf{W}_{u\,j}\big)^{-1} \big(\tilde{\mathbf{P}}^T \mathbf{W}_{e\,j} \tilde{\mathbf{P}} + \mathbf{W}_{\Delta u\,j}\big), \tag{1.47}$$

	Design procedure
Step 1.	Describes a lifted form representation of the nominal model describing the relation between the ILC input and the resulting output of the system.
Step 2.	Select weighting matrices $\mathbf{W}_{e\,j}$, $\mathbf{W}_{\Delta u\,j}$ and $\mathbf{W}_{u\,j}$.
Step 3.	Design the input update by minimising a quadratic function in the error and the control signal. The resulting update law depend on two matrices that can be interpreted as matrices $\mathbf{Q}_{opt}$ and $\mathbf{L}_{opt}$.

Table 1.3: A Q-ILC design algorithm

and where the optimal learning function is:

$$\mathbf{L}_{opt} = \left(\tilde{\mathbf{P}}^T \mathbf{W}_{e\,j} \tilde{\mathbf{P}} + \mathbf{W}_{\Delta u\,j} + \mathbf{W}_{u\,j}\right)^{-1} \tilde{\mathbf{P}}^T \mathbf{W}_{e\,j}. \tag{1.48}$$

The design algorithm is summarised in Table 1.3.

Performance, convergent rate and robustness of the algorithm can be modified by choosing appropriate weighting matrices. Specifically, the weighting term $\mathbf{W}_{\Delta u\,j}$ affects the convergence speed, that is how quickly the input changes form one iteration to the next, but has no effect on the asymptotic error; whereas the weight $\mathbf{W}_{u\,j}$, that is useful to limit the control action and prevent saturation, influence the final tracking error.

In the next chapter a new model-based ILC method, known as Kalman-ILC, is presented. This new design results in a Q-ILC when specific conditions are met.

Multi-Objective ILC design methods Multi-Objective ILC ([14] [15]) are new design methods that use an optimization problem [11], reformulated as a convex problem, to trade different objectives (introduced also as constraints), by solving simultaneously the non-causal Q-filter and learning function. An example, of the procedure of these methods is the following:

$$\begin{aligned} \underset{Q(q),L(q)}{\text{minimize}} \quad & \text{convergence speed} \\ \text{subject to} \quad & \text{robust convergence} \\ & \text{robust performance} \\ & \text{input constraint} \end{aligned} \tag{1.49}$$

where the solutions are the two filters of the general ILC input update equation (1.14).

[11]These design methods can belong, also, to the category of frequency-domain methods

Chapter 2

Kalman Iterative Learning Control

This chapter is centred on the main topic of this thesis: the Kalman iterative learning control. It is organized as follows: first, an introduction to the reasons of K-ILC design choice is discussed, then, the system description and its lifted-domain representation are presented. Later on, the Kalman-ILC algorithm is outlined and the estimation step (Kalamn filter) and input update step (optimization problem) are explained in detail. The chapter ends with an analysis of the design parameter choices and a comparison between the K-ILC and the Q-ILC.

2.1 Introduction to estimation based ILC

The previous chapter concludes with an analysis of the ILC design methods commonly used, where a major distinction is done between basic and model-based design techniques. As explained, the model-based designs are preferred to basic ones in order to achieve the best results in terms of system performance, even at the expense of increasing the problem complexity (and so the computational complexity).

On top of that, a *serial* arrangement scheme is considered. In fact, the purpose of this thesis is to implement an ILC to a pre-defined quadcopter model, without interfering with its inner dynamics. In this way, it is possible to change the performance of the UAV by modifying the setpoint only, without altering the feedback control laws of the system.

Traditionally, ILC has been applied to systems where the controlled variables are also the measured variables (this is specified in Arimoto's postulate **P5** in 1.2). When this is not the case, as in the problem studied, although the ILC algorithm reduces the error of measured variables, the performance, evaluated in terms of controlled variables, can be worsened due to model errors.

For this reason, ILC model-based algorithms have been extended with estimators, applied to iteration-domain variables, in order to improve the learning performance. This has proven to be particularly effective when a more accurate model of the dynamics of the system is available and when non-repetitive noise perturbations cannot be neglected. Through the estimation step, the unknown repetitive disturbance, that represents the discrepancy between the nominal model and actual model, can be estimated by measuring the output only (if the model of the system is already defined). This information is exploited and used to update the ILC input.

Within the model-based *serial* arrangement ILC approaches, optimization based ILC design methods, also called *norm-optimal*, are selected. As discussed in the prior chapter, these algorithms compute the input update step solving an optimization problem (subject to additional constraints), by minimizing the predicted tracking error of the next iteration.

The model-based, norm-optimal, estimation-enhanced ILC addressed in this thesis and discussed in this chapter is the Kalman filter-enhanced iterative learning control [4] [16], K-ILC for short.

2.2 System description

Before introducing the K-ILC, the (feedback) system, in which the control is implemented, is described, and its lifted form representation is derived. The pros in using a lifted representation were already commented in the previous chapter. In this section, the lifted description is obtained directly from the state-space form, both for linear time-invariant and time-variant systems.

2.2.1 Model of dynamics

The model that capture the dynamics of the physical system under consideration is assumed to be given. As the natural domain of ILC is the discrete-domain, the system is represented in the discrete time-domain. The MIMO model is described by a set of linear strictly proper differential equations defined in the state-space form:

$$\begin{cases} \tilde{\mathbf{x}}(k+1) = \mathbf{A}_d\tilde{\mathbf{x}}(k) + \mathbf{B}_d\tilde{\mathbf{u}}(k) \\ \tilde{\mathbf{y}}(k+1) = \mathbf{C}_d\tilde{\mathbf{x}}(k+1), \end{cases} \tag{2.1}$$

where $k \in \mathcal{K}$ is the finite discrete-time index, $\mathcal{K} := \{0, 1, \dots, N-1\}$ and $N < \infty$ is the finite trial length. $\tilde{\mathbf{x}}(k) \in \mathbb{R}^{n_x}$, $\tilde{\mathbf{u}}(k) \in \mathbb{R}^{n_u}$ and $\tilde{\mathbf{y}}(k) \in \mathbb{R}^{n_y}$ represent respectively the vector of the states, the vector of the inputs and the one of the outputs according to the model approximation of the physical system. The initial condition is $\tilde{\mathbf{x}}(0) = \tilde{\mathbf{x}}_0$ and, as the system accommodate the one-step delay, it is also defined the output at the initial instant, that is $\tilde{\mathbf{y}}(0) = \mathbf{C}_d\tilde{\mathbf{x}}_0$.

For the sake of simplicity the subscript 'd' will be dropped from the notation when referring to the LTI system matrices.

The lifted form of this model is exploited to derive a static map that capture the system dynamics during one trial.

2.2.2 Lifted representation

Linear time-invariant system

A lifted representation is given for the discrete strictly proper LTI system (2.1).

The state and output motions of the system starting from the initial conditions defined at the instant $k = 0$, which is equal to $\tilde{\mathbf{x}}(0) = \tilde{\mathbf{x}}_0$ (and with $\tilde{\mathbf{y}}(0) = \mathbf{C}\tilde{\mathbf{x}}_0$), is:

$$\begin{array}{rcl}
\tilde{\mathbf{x}}(1) &=& \mathbf{A}\tilde{\mathbf{x}}(0) + \mathbf{B}\tilde{\mathbf{u}}(0) = \mathbf{A}\tilde{\mathbf{x}}_0 + \mathbf{B}\tilde{\mathbf{u}}(0) \\
\tilde{\mathbf{x}}(2) &=& \mathbf{A}\tilde{\mathbf{x}}(1) + \mathbf{B}\tilde{\mathbf{u}}(1) = \mathbf{A}^2\tilde{\mathbf{x}}_0 + \mathbf{A}\mathbf{B}\tilde{\mathbf{u}}(0) + \mathbf{B}\tilde{\mathbf{u}}(1) \\
\tilde{\mathbf{x}}(3) &=& \mathbf{A}\tilde{\mathbf{x}}(2) + \mathbf{B}\tilde{\mathbf{u}}(2) = \mathbf{A}^3\tilde{\mathbf{x}}_0 + \mathbf{A}^2\mathbf{B}\tilde{\mathbf{u}}(0) + \mathbf{A}\mathbf{B}\tilde{\mathbf{u}}(1) + \mathbf{B}\tilde{\mathbf{u}}(2) \\
&\vdots& \\
\tilde{\mathbf{x}}(N) &=& \mathbf{A}\tilde{\mathbf{x}}(N-1) + \mathbf{B}\tilde{\mathbf{u}}(N-1) = \mathbf{A}^N\tilde{\mathbf{x}}_0 + \sum_{i=0}^{k-1}\left[\mathbf{A}^{N-i-1}\mathbf{B}\tilde{\mathbf{u}}(i)\right] \\
\\
\tilde{\mathbf{y}}(1) &=& \mathbf{C}\tilde{\mathbf{x}}(1) = \mathbf{C}\mathbf{A}\tilde{\mathbf{x}}_0 + \mathbf{C}\mathbf{B}\tilde{\mathbf{u}}(0) \\
\tilde{\mathbf{y}}(2) &=& \mathbf{C}\tilde{\mathbf{x}}(2) = \mathbf{C}\mathbf{A}^2\tilde{\mathbf{x}}_0 + \mathbf{C}\mathbf{A}\mathbf{B}\tilde{\mathbf{u}}(0) + \mathbf{C}\mathbf{B}\tilde{\mathbf{u}}(1) \\
\tilde{\mathbf{y}}(3) &=& \mathbf{C}\tilde{\mathbf{x}}(3) = \mathbf{C}\mathbf{A}^3\tilde{\mathbf{x}}_0 + \mathbf{C}\mathbf{A}^2\mathbf{B}\tilde{\mathbf{u}}(0) + \mathbf{C}\mathbf{A}\mathbf{B}\tilde{\mathbf{u}}(1) + \mathbf{C}\mathbf{B}\tilde{\mathbf{u}}(2) \\
&\vdots& \\
\tilde{\mathbf{y}}(N) &=& \mathbf{C}\tilde{\mathbf{x}}(N) = \mathbf{C}\mathbf{A}^N\tilde{\mathbf{x}}_0 + \sum_{i=0}^{k-1}\left[\mathbf{C}\mathbf{A}^{N-i-1}\mathbf{B}\tilde{\mathbf{u}}(i)\right].
\end{array}$$

The state and output dynamic can be rewritten in matrix notation:

$$\begin{bmatrix}\tilde{\mathbf{x}}(1)\\ \tilde{\mathbf{x}}(2)\\ \tilde{\mathbf{x}}(3)\\ \vdots\\ \tilde{\mathbf{x}}(N)\end{bmatrix} = \begin{bmatrix}\mathbf{B} & \mathbf{0} & \cdots & \cdots & \mathbf{0}\\ \mathbf{A}\mathbf{B} & \mathbf{B} & \ddots & & \vdots\\ \mathbf{A}^2\mathbf{B} & \mathbf{A}\mathbf{B} & \mathbf{B} & \ddots & \vdots\\ \vdots & \ddots & \ddots & \ddots & \mathbf{0}\\ \mathbf{A}^{N-1}\mathbf{B} & \cdots & \mathbf{A}^2\mathbf{B} & \mathbf{A}\mathbf{B} & \mathbf{B}\end{bmatrix}\begin{bmatrix}\tilde{\mathbf{u}}(0)\\ \tilde{\mathbf{u}}(1)\\ \tilde{\mathbf{u}}(2)\\ \vdots\\ \tilde{\mathbf{u}}(N-1)\end{bmatrix} + \begin{bmatrix}\mathbf{A}\tilde{\mathbf{x}}_0\\ \mathbf{A}^2\tilde{\mathbf{x}}_0\\ \mathbf{A}^3\tilde{\mathbf{x}}_0\\ \vdots\\ \mathbf{A}^N\tilde{\mathbf{x}}_0\end{bmatrix} \tag{2.2}$$

$$\begin{bmatrix}\tilde{\mathbf{y}}(1)\\ \tilde{\mathbf{y}}(2)\\ \vdots\\ \tilde{\mathbf{y}}(N)\end{bmatrix} = \begin{bmatrix}\mathbf{C} & \mathbf{0} & \cdots & \mathbf{0}\\ \mathbf{0} & \mathbf{C} & \ddots & \vdots\\ \vdots & \ddots & \ddots & \mathbf{0}\\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{C}\end{bmatrix}\begin{bmatrix}\tilde{\mathbf{x}}(1)\\ \tilde{\mathbf{x}}(2)\\ \vdots\\ \tilde{\mathbf{x}}(N)\end{bmatrix} \tag{2.3}$$

The lifted representation of the discrete-time signals $\tilde{\mathbf{u}}(k)$, $\tilde{\mathbf{x}}(k)$ and $\tilde{\mathbf{y}}(k)$ are named respectively $\tilde{\mathbf{U}}$, $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$, in order to be consistent in defining nominal variables, *i.e.* modeled variables, with tilde and lifted variables with capital letters[1]. Defining:

$$
\begin{aligned}
\tilde{\mathbf{U}} &:= \left[\tilde{\mathbf{u}}(0), \tilde{\mathbf{u}}(1), \ldots, \tilde{\mathbf{u}}(N-1)\right]^T, \quad \tilde{\mathbf{U}} \in \mathbb{R}^{Nn_u},\\
\tilde{\mathbf{X}} &:= \left[\tilde{\mathbf{x}}(1), \tilde{\mathbf{x}}(2), \ldots, \tilde{\mathbf{x}}(N)\right]^T, \quad \tilde{\mathbf{X}} \in \mathbb{R}^{Nn_x},\\
\tilde{\mathbf{Y}} &:= \left[\tilde{\mathbf{y}}(1), \tilde{\mathbf{y}}(2), \ldots, \tilde{\mathbf{y}}(N)\right]^T, \quad \tilde{\mathbf{Y}} \in \mathbb{R}^{Nn_y},\\
\tilde{\mathbf{F}} &:= \begin{bmatrix} \mathbf{B} & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ \mathbf{AB} & \mathbf{B} & \ddots & & \vdots \\ \mathbf{A}^2\mathbf{B} & \mathbf{AB} & \mathbf{B} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \mathbf{0} \\ \mathbf{A}^{N-1}\mathbf{B} & \cdots & \mathbf{A}^2\mathbf{B} & \mathbf{AB} & \mathbf{B} \end{bmatrix}, \quad \tilde{\mathbf{F}} \in \mathbb{R}^{Nn_x \times Nn_u},\\
\tilde{\mathbf{G}} &:= \begin{bmatrix} \mathbf{C} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{C} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{C} \end{bmatrix}, \quad \tilde{\mathbf{G}} \in \mathbb{R}^{Nn_y \times Nn_x},\\
\mathbf{\Delta}^0 &:= \left[\left(\mathbf{A}\tilde{\mathbf{x}}_0\right)^T \ \left(\mathbf{A}^2\tilde{\mathbf{x}}_0\right)^T \ \left(\mathbf{A}^3\tilde{\mathbf{x}}_0\right)^T \ \cdots \ \left(\mathbf{A}^N\tilde{\mathbf{x}}_0\right)^T\right]_T, \quad \mathbf{\Delta}^0 \in \mathbb{R}^{Nn_x},
\end{aligned}
$$

equations (2.2) and (2.3) can be written in short in the following system:

$$
\begin{cases} \tilde{\mathbf{X}} = \tilde{\mathbf{F}}\tilde{\mathbf{U}} + \mathbf{\Delta}^0 \\ \tilde{\mathbf{Y}} = \tilde{\mathbf{G}}\tilde{\mathbf{X}} \end{cases} \tag{2.4}
$$

where the vector $\mathbf{\Delta}^0$ represent the state disturbance of the system due to non-null initial conditions.

Linear time-varying system

A similar description can be used also to represent in lifted form a discrete strictly proper Linear Time-Varying (LTV) system. If the model that represents the dynamics of the physical system is so defined:

$$
\begin{cases} \tilde{\mathbf{x}}(k+1) = \mathbf{A}(k)\tilde{\mathbf{x}}(k) + \mathbf{B}(k)\tilde{\mathbf{u}}(k) \\ \tilde{\mathbf{y}}(k+1) = \mathbf{C}(k+1)\tilde{\mathbf{x}}(k+1), \end{cases} \tag{2.5}
$$

the LTV system can be rewritten in lifted form as (2.4).

[1]The notation is different from the one used in the previous chapter. From here on the lifted vector are written in capital letters.

The state and output motions of the system with initial condition $\tilde{\mathbf{x}}(0) = \tilde{\mathbf{x}}_0$, and one step delay (with $\tilde{\mathbf{y}}(0) = \mathbf{C}(0)\tilde{\mathbf{x}}_0$) are:

$$\begin{array}{rcl}
\tilde{\mathbf{x}}(1) & = & \mathbf{A}(0)\tilde{\mathbf{x}}(0) + \mathbf{B}(0)\tilde{\mathbf{u}}(0) = \mathbf{A}(0)\tilde{\mathbf{x}}_0 + \mathbf{B}(0)\tilde{\mathbf{u}}(0) \\
\tilde{\mathbf{x}}(2) & = & \mathbf{A}(1)\tilde{\mathbf{x}}(1) + \mathbf{B}(1)\tilde{\mathbf{u}}(1) = \mathbf{A}(1)\mathbf{A}(0)\tilde{\mathbf{x}}_0 + \mathbf{A}(1)\mathbf{B}(0)\tilde{\mathbf{u}}(0) + \mathbf{B}(1)\tilde{\mathbf{u}}(1) \\
\tilde{\mathbf{x}}(3) & = & \mathbf{A}(2)\tilde{\mathbf{x}}(2) + \mathbf{B}(2)\tilde{\mathbf{u}}(2) = \mathbf{A}(2)\mathbf{A}(1)\mathbf{A}(0)\tilde{\mathbf{x}}_0 + \mathbf{A}(2)\mathbf{A}(1)\mathbf{B}(0)\tilde{\mathbf{u}}(0) \\
& & \qquad +\mathbf{A}(2)\mathbf{B}(1)\tilde{\mathbf{u}}(1) + \mathbf{B}(2)\tilde{\mathbf{u}}(2) \\
& \vdots & \\
\tilde{\mathbf{x}}(N) & = & \mathbf{A}(N-1)\tilde{\mathbf{x}}(N-1) + \mathbf{B}(N-1)\tilde{\mathbf{u}}(N-1) = \left(\prod_{i=0}^{N-1} \mathbf{A}(i)\right)\tilde{\mathbf{x}}_0 \\
& & \qquad + \sum_{m=0}^{N-1} \left[\left(\prod_{n=0}^{N-m-2} \mathbf{A}(N-n-1)\right)\mathbf{B}(m)\tilde{\mathbf{u}}(m)\right] \\
\tilde{\mathbf{y}}(1) & = & \mathbf{C}(1)\tilde{\mathbf{x}}(1) = \mathbf{C}(1)\mathbf{A}(0)\tilde{\mathbf{x}}_0 + \mathbf{C}(1)\mathbf{B}(0)\tilde{\mathbf{u}}(0) \\
\tilde{\mathbf{y}}(2) & = & \mathbf{C}(2)\tilde{\mathbf{x}}(2) = \mathbf{C}(2)\mathbf{A}(1)\mathbf{A}(0)\tilde{\mathbf{x}}_0 + \mathbf{C}(2)\mathbf{A}(1)\mathbf{B}(0)\tilde{\mathbf{u}}(0) \\
& & \qquad +\mathbf{C}(2)\mathbf{B}(1)\tilde{\mathbf{u}}(1) \\
\tilde{\mathbf{y}}(3) & = & \mathbf{C}(3)\tilde{\mathbf{x}}(3) = \mathbf{C}(3)\mathbf{A}(2)\mathbf{A}(1)\mathbf{A}(0)\tilde{\mathbf{x}}_0 + \mathbf{C}(3)\mathbf{A}(2)\mathbf{A}(1)\mathbf{B}(0)\tilde{\mathbf{u}}(0) \\
& & \qquad +\mathbf{C}(3)\mathbf{A}(2)\mathbf{B}(1)\tilde{\mathbf{u}}(1) + \mathbf{C}(3)\mathbf{B}(2)\tilde{\mathbf{u}}(2) \\
& \vdots & \\
\tilde{\mathbf{y}}(N) & = & \mathbf{C}(N)\tilde{\mathbf{x}}(N) = \mathbf{C}(N)\left(\prod_{i=0}^{N-1} \mathbf{A}(i)\right)\tilde{\mathbf{x}}_0 \\
& & \qquad + \sum_{m=0}^{N-1} \left[\mathbf{C}(N)\left(\prod_{n=0}^{N-m-2} \mathbf{A}(N-n-1)\right)\mathbf{B}(m)\tilde{\mathbf{u}}(m)\right].
\end{array}$$

In this case matrices $\tilde{\mathbf{F}}$ and $\tilde{\mathbf{G}}$, and vector $\mathbf{\Delta}^0$ of system (2.4) are defined as follows.

- Matrix $\tilde{\mathbf{F}}$, indicated with two subscripts identifying the initial and final instants of the time sequence to which they refer, is:

$$\tilde{\mathbf{F}}_{0,N} = \begin{bmatrix} \tilde{\mathbf{F}}_{(1,1)} & \cdots & \tilde{\mathbf{F}}_{(1,N)} \\ \vdots & \ddots & \vdots \\ \tilde{\mathbf{F}}_{(N,1)} & \cdots & \tilde{\mathbf{F}}_{(N,N)} \end{bmatrix}, \tag{2.6}$$

  with $\tilde{\mathbf{F}}_{0,N} \in \mathbb{R}^{Nn_x \times Nn_u}$ and with $\tilde{\mathbf{F}}_{(l,m)} \in \mathbb{R}^{n_x \times n_u}$ equal to:

$$\tilde{\mathbf{F}}_{(l,m)} = \begin{cases} \mathbf{A}(l-1)\cdots\mathbf{A}(m)\mathbf{B}(m-1) & \text{if } m < l \\ \mathbf{B}(m-1) & \text{if } m = l \\ 0 & \text{if } m > l, \end{cases} \tag{2.7}$$

  where $1 \le l, m \le N$.

- Matrix $\tilde{\mathbf{G}}$ is defined as:

$$\tilde{\mathbf{G}}_{0,N} = \begin{bmatrix} \tilde{\mathbf{G}}_{(1,1)} & \cdots & \tilde{\mathbf{G}}_{(1,N)} \\ \vdots & \ddots & \vdots \\ \tilde{\mathbf{G}}_{(N,1)} & \cdots & \tilde{\mathbf{G}}_{(N,N)} \end{bmatrix}, \tag{2.8}$$

with $\tilde{\mathbf{G}}_{0,N} \in \mathbb{R}^{Nn_y \times Nn_x}$ and with $\tilde{\mathbf{G}}_{(l,m)} \in \mathbb{R}^{n_y \times n_x}$ equal to:

$$\tilde{\mathbf{G}}_{(l,m)} = \begin{cases} \mathbf{C}(l) & \text{if } m = l \\ 0 & \text{if } m \neq l, \end{cases} \tag{2.9}$$

analogously with $1 \leq l, m \leq N$.

- Vector $\mathbf{\Delta}^0$ is instead:

$$\mathbf{\Delta}^0 = \left[\left(\mathbf{A}(0)\tilde{\mathbf{x}}_0 \right)^T \ \left(\mathbf{A}(1)\mathbf{A}(0)\tilde{\mathbf{x}}_0 \right)^T, \ \ldots, \ \left(\prod_{i=0}^{N-1} \mathbf{A}(i)\tilde{\mathbf{x}}_0 \right)^T \right]^T, \tag{2.10}$$

with: $\mathbf{\Delta}^0 \in \mathbb{R}^{Nn_x}$ and with $\tilde{\mathbf{x}}(0) = \tilde{\mathbf{x}}_0 \in \mathbb{R}^{n_x}$.

Nominal lifted model

The nominal model lifted matrix $\mathbf{F}$[2] linking the input $\tilde{\mathbf{U}}$ to the output $\tilde{\mathbf{Y}}$ can be found solving the system (2.4) rewritten below:

$$\begin{cases} \tilde{\mathbf{X}} = \tilde{\mathbf{F}}\tilde{\mathbf{U}} + \mathbf{\Delta}^0 \\ \tilde{\mathbf{Y}} = \tilde{\mathbf{G}}\tilde{\mathbf{X}}, \end{cases} \tag{2.11}$$

that is the lifted model of the linear system (2.1) (or (2.5) for the time-variant case); the output turns out to be:

$$\tilde{\mathbf{Y}} = \mathbf{F}\,\tilde{\mathbf{U}} + \mathbf{D}^0, \tag{2.12}$$

with:

$$\mathbf{F} = \tilde{\mathbf{G}}\tilde{\mathbf{F}}, \quad \in \mathbb{R}^{Nn_y \times Nn_u}, \tag{2.13}$$

$$\mathbf{D}^0 = \tilde{\mathbf{G}}\mathbf{\Delta}^0, \quad \in \mathbb{R}^{Nn_y}. \tag{2.14}$$

The nominal model describing the multi-input/multi-output relation, is redefined with null initial conditions:

$$\tilde{\mathbf{Y}} = \mathbf{F}\,\tilde{\mathbf{U}}. \tag{2.15}$$

The nominal model lifted representation is, in the end, a static linear mapping: it captures the complete time-domain dynamics of the system by mapping the finite input discrete-time series $\tilde{\mathbf{u}}(k)$, $k \in \mathcal{K}$, onto the one-step shifted output time series $\tilde{\mathbf{y}}(k+1)$, $k \in \mathcal{K}$[3].

[2]Matrix $\mathbf{F}$ corresponds to what in the previous chapter was indicated with $\tilde{\mathbf{P}}$, *i.e.* the lifted nominal model of the plant. The letter 'F' indicates the fact that the system in question is a (stable) feedback system.

[3]$\tilde{\mathbf{u}}(k)$ is the input vector that enters the system (2.1) while $\tilde{\mathbf{y}}(k+1)$ is the corresponding output; they are distinguished from their lifted form counterparts: $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{Y}}$.

2.2.3 ILC system in lifted form

The objective of the Kalamn iterative learning control, as any other *serial* ILC schemes, is to improve the performance of a system in executing the same task or trajectory by updating iteratively the input of the system (which corresponds to the output of the iterative learning controller) after each trial.

Nominal and 'actual' ILC model

The lifted representation is used to define the dynamics of the K-ILC system. As seen before, the evolution of the system dynamics in the lifted-domain over one trial is, in fact, modeled by the following nominal lifted system $\mathcal{S}_n$[4]:

$$\mathcal{S}_n : \begin{cases} \tilde{\mathbf{X}} = \tilde{\mathbf{F}}\tilde{\mathbf{U}} \\ \tilde{\mathbf{Y}} = \tilde{\mathbf{G}}\tilde{\mathbf{X}}, \end{cases} \tag{2.16}$$

or equivalently:

$$\tilde{\mathbf{Y}} = \mathbf{F}\tilde{\mathbf{U}}. \tag{2.17}$$

$\boldsymbol{\Delta}_j^0$, which describes the state motion of non-zeros initial conditions, is omitted from the formulation as it is assumed to be null since non-zero initial conditions are accounted as disturbances in the generalized model.

The tilde is used so far to indicate the variables (namely $\tilde{\mathbf{U}}$, $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$) of the ideal model that approximates the dynamical system over a single trial. This approximating nominal lifted model does not account for model uncertainties, repeated disturbances and process and measurement noises. Those are considered in the generalized model $\mathcal{S}$ which describes the system behaviour over multiple iterations:

$$\mathcal{S} : \begin{cases} \mathbf{X}_j = \tilde{\mathbf{F}}\mathbf{U}_j + \boldsymbol{\Delta}_j + \mathbf{N}_\xi \boldsymbol{\Xi}_j \\ \mathbf{Y}_j = \tilde{\mathbf{G}}\mathbf{X}_j + \mathbf{N}_\lambda \boldsymbol{\Lambda}_j. \end{cases} \tag{2.18}$$

These equations differ from the ones of the system $\mathcal{S}_n$ by the subscript j, with $j \in \mathcal{J} := \{0, 1, \dots, N_j\}$, indicating the j-th trial execution of the desired task. The nominal vectors $\tilde{\mathbf{X}}$, $\tilde{\mathbf{U}}$, $\tilde{\mathbf{Y}}$ are replaced with the 'actual' vectors $\mathbf{X}_j$, $\mathbf{U}_j$, $\mathbf{Y}_j$. The term $\boldsymbol{\Delta}_j$ represents the repetitive state disturbance, which captures repetitive disturbances, model errors and repeated non-zero initial conditions ($\boldsymbol{\Delta}_j^0$). It is subject to minimal changes between iterations, in fact, the state disturbance from one iteration to the next differs only for a contribution of a trial-uncorrelated sequence of zero-mean Gaussian noise.

The signals $\boldsymbol{\Xi}_j$ and $\boldsymbol{\Lambda}_j$ account, instead, for process and measurement noises. They are assumed to be trial-uncorrelated sequences of zero-mean Gaussian noises too.

[4]The system is the lifted form representation of the system (2.1) when the initial condition is null

The system (2.18) is solved as follows:

$$\mathbf{Y}_j = \tilde{\mathbf{G}}\tilde{\mathbf{F}}\mathbf{U}_j + \tilde{\mathbf{G}}\boldsymbol{\Delta}_j + \tilde{\mathbf{G}}\mathbf{N}_\xi\boldsymbol{\Xi}_j + \mathbf{N}_\lambda\boldsymbol{\Lambda}_j, \tag{2.19}$$

and condensed in the equation:

$$\mathbf{Y}_j = \mathbf{F}\mathbf{U}_j + \mathbf{D}_j + \mathbf{M}_j, \tag{2.20}$$

linking the input vector $\mathbf{U}_j$ to the measured output vector $\mathbf{Y}_j$.

The vector $\mathbf{D}_j$ represents the repetitive disturbance which captures the tracking errors due to unmodeled dynamics, unmodeled external noise and disturbances. It derives from the state disturbance $\boldsymbol{\Delta}_j$ and it is so defined:

$$\mathbf{D}_{j+1} = \mathbf{D}_j + \boldsymbol{\Omega}_j. \tag{2.21}$$

As for the state disturbance $\boldsymbol{\Delta}$, the disturbance $\mathbf{D}$ enclosed the repetitive behaviour of the outputs so to differ, from an iteration to the next, for a random signal trial-dependent $\boldsymbol{\Omega}_j$, modeled as a zero-mean Gaussian noise with covariance $\boldsymbol{\Sigma}_j$.

The last term $\mathbf{M}_j$ is also a zero-mean Gaussian trial-dependent noise with covariance matrix $\mathbf{H}_j$ and is equal to: $\mathbf{M}_j = \tilde{\mathbf{G}}\mathbf{N}_\xi\boldsymbol{\Xi}_j + \mathbf{N}_\lambda\boldsymbol{\Lambda}_j$.

ILC tracking error

Specifically, the goal of the ILC is to to track a desired output vector:

$$\mathbf{Y}_d = \left[\tilde{\mathbf{y}}_d(1), \tilde{\mathbf{y}}_d(2), \ldots, \tilde{\mathbf{y}}_d(N)\right]^T, \quad \mathbf{Y}_d \in \mathbb{R}^{Nn_y}, \tag{2.22}$$

where $\tilde{\mathbf{y}}_d(k)$ describes the desired output signal over time (in the discrete-domain). To meet the objective, the tracking error defined as[5]:

$$\mathbf{E}_j = \mathbf{Y}_j - \mathbf{Y}_d, \tag{2.23}$$

needs to decrease during iterations through a proper action on the input vector $\mathbf{U}_j$. A new vector $\mathbf{U}_{nom}$ is introduced in the equation below:

$$\mathbf{Y}_d = \mathbf{F}\mathbf{U}_{nom}, \tag{2.24}$$

in order to represents the nominal input that results in the desired output, in the ideal case in which the nominal model is perfect.

In the ideal case in which the nominal system (2.17) is perfectly suited to describe the dynamics of the system during the different iterations, *i.e.*, when model uncertainties, repeated disturbances and process and measurement noises of the 'actual' model are neglected, the nominal output is equal to:

$$\tilde{\mathbf{Y}}_j = \mathbf{F}\mathbf{U}_j, \tag{2.25}$$

[5]The (lifted) error definition differs to the one given in the prior chapter only for the sign.

with $\tilde{\mathbf{U}}_j = \mathbf{U}_j$, can be used to define a nominal iterative tracking error:

$$\tilde{\mathbf{E}}_j = \tilde{\mathbf{Y}}_j - \mathbf{Y}_d, \tag{2.26}$$

where $\tilde{\mathbf{Y}}_j$ is the output obtained form

Replacing the nominal output and the desired reference definitions (equations (2.25) and (2.24)), the nominal iterative tracking error becomes:

$$\tilde{\mathbf{E}}_j = \mathbf{F}\mathbf{V}_j, \tag{2.27}$$

with $\mathbf{V}_j := \mathbf{U}_j - \mathbf{U}_{nom}$.

By substituting the actual output equation (2.20) in (2.23), the 'real' tracking error becomes:

$$\mathbf{E}_j = \mathbf{F}\mathbf{V}_j + \mathbf{D}_j + \mathbf{M}_j, \tag{2.28}$$

with $\mathbf{M}_j \sim \mathcal{N}(0, \mathbf{H}_j)$.

2.3 K-ILC

After having introduced the whole theoretical description of a typical ILC in *serial* arrangement in lifted form, the Kalman filter enhanced ILC (K-ILC) is now explained in depth.

The Kalman-ILC is first of all a norm-optimal ILC. As such, K-ILC uses information of the input and the error at the current trial, $\mathbf{U}_j$ and $\mathbf{E}_j$, to design the next trial input, $\mathbf{U}_{j+1}$, that minimizes the tracking error $\mathbf{E}_{j+1}$, *i.e.*, the discrepancy between the actual and the desired output at the upcoming iteration. This ILC input update is generated by solving an optimization problem (with possible constraints in the input), and, since it enters the system as an input, it does not change the inner dynamics of the system.

A generic scheme of a first-order norm-optimal ILC in lifted form is illustrated in Figure 2.1.

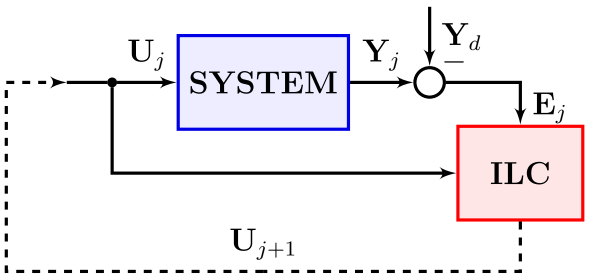


Figure 2.1: Norm-optimal ILC block diagram in lifted form

Note that for the sake of clarity, the variables displayed in Figure 2.1 are in lifted form ($\mathbf{U}_j$, $\mathbf{Y}_j$, $\mathbf{Y}_d$, $\mathbf{E}_j$). Actually, the system works in the discrete-time-

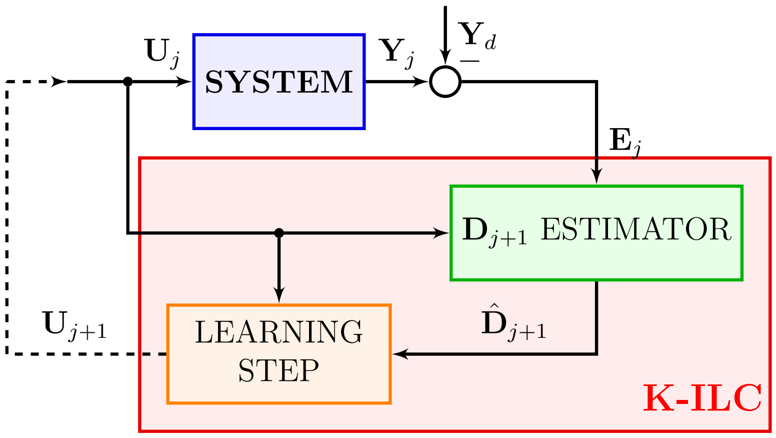


Figure 2.2: K-ILC block diagram in lifted form

domain with discrete-time signals ($\mathbf{u}_j(k)$, $\mathbf{y}_j(k)$, $\mathbf{y}_d(k)$, $\mathbf{e}_j(k)$[6]). The iterative learning controller is indeed responsible for storing these discrete signals and retrieving it in the lifted form description.

The Kalman filter enhances the norm-optimal ILC by introducing an additional estimator that estimates the disturbance vector of the next iteration disturbance $\mathbf{D}_{j+1}$, based on the current input $\mathbf{U}_j$ and error measurement $\mathbf{E}_j$. The term $\mathbf{D}$ is once again the disturbance vector that represents the difference between the nominal model and the real one (as well as the difference between the nominal error (2.2.3) and the real error (2.28)), aside from the measurement noise term $\mathbf{M}_j$. After the estimation step is completed, the learning step updates the next trial input. This is done by solving the usual optimization problem that is design precisely for eliminating the estimated repetitive undesired disturbance.

The general scheme is displayed in Figure 2.2.

The disturbance estimate $\hat{\mathbf{D}}_{j+1}$ is obtained from a Kalman filter based on the stochastic model below, obtained joining equations (2.21) and (2.28):

$$\mathcal{S}_{\mathcal{K}} : \begin{cases} \mathbf{D}_{j+1} = \mathbf{D}_j + \mathbf{\Omega}_j \\ \mathbf{E}_j = \mathbf{F}\mathbf{V}_j + \mathbf{D}_j + \mathbf{M}_j. \end{cases} \tag{2.29}$$

The terms $\mathbf{\Omega}_j$ and $\mathbf{M}_j$ are two random vectors (both with n_y entries) representing the noise acting on the state and output equations, respectively. These terms are described as Gaussian noises with zero mean values and given variance matrices, $\mathbf{\Sigma}_j$ and $\mathbf{H}_j$:

$$\boldsymbol{\omega}_j \sim \mathcal{N}(0, \mathbf{\Sigma}_j), \tag{2.30}$$
$$\boldsymbol{\mu}_j \sim \mathcal{N}(0, \mathbf{H}_j). \tag{2.31}$$

[6]Recall the notation used: $\mathbf{u}_j$ is the lifted form vector of the discrete-time signal $\mathbf{u}_j(k)$; the same is true for the other variables.

In the proposed model, the disturbance $\mathbf{D}_j$ represents the state of the dynamical system (2.29), while the error $\mathbf{E}_j$ defines the output measurements. The aim of Kalman filter prediction step is to find a recursive expression for the optimal estimate of the state at the iteration $j{+}1$: $\mathbf{D}_{j+1}$, given the error observations up to iteration j: $\hat{\mathbf{D}}_{j+1|j}$.

The next iteration disturbance and error estimates are defined as:

$$\begin{aligned}\hat{\mathbf{D}}_{j+1|j} &:= E[\,\mathbf{D}_{j+1}\,|\,\mathbf{E}^j\,],\\ \hat{\mathbf{E}}_{j+1|j} &:= E[\,\mathbf{E}_{j+1}\,|\,\mathbf{E}^j\,],\end{aligned}$$

where: the ‘hat’ refers to estimated variables, E denotes the expectation operator of a random variable and $\mathbf{E}^j$ is the vector that groups together the set of error data over the interval of iterations $\mathcal{J}^- := \{\, j, j-1, \dots, 0\,\}$:

$$\mathbf{E}^j = [\,\mathbf{E}_j^T, \mathbf{E}_{j-1}^T, \dots, \mathbf{E}_0^T\,]^T. \tag{2.32}$$

The vector $\mathbf{E}^j$ generates a subspace known as the *subspace of the past.*

The implementation of the actual Kalman estimator is developed in the next section, while the setup of the optimization problem for the input update is discussed soon after.

2.4 Kalman estimator

The Kalman estimator is an optimal mathematical tool extensively used in the field of statistical estimation. The theory presented in this section is known as *Kalman filtering* because it originates from the contributions of R.E. Kalman.

Kalman filtering addresses a general problem arising when the unknown variable, the state, may be different from the observed variable, the output. The estimation of the state is performed by exploiting the information of it hidden in the observed output. The link between the state and the output is expressed by the mathematical state space model representing the system dynamics.

This algorithm is appropriate to estimate the disturbance of the linear dynamic system (2.29), whose state is perturbed (in the iteration-domain) by a noise, by using measurements which are also linear with respect to the state and corrupted by noise. The iteration-domain Kalman filter detailed, in fact, uses a series of measurements observed over the iterations, disturbed by noise and possibly other inaccuracies, to produce the best estimates of unknown variables. These tend to be more accurate than those based on a single measurement alone, because, for each iteration, it is estimated a joint probability distribution of the variables in order to minimize the covariance of the state prediction error. Optimality of the filter assumes that the noises in equation (2.33) and (2.34) are white Gaussian noises:

$$\mathbf{\Omega}_j \sim \mathcal{WN}(0, \mathbf{\Sigma}_j), \tag{2.33}$$
$$\mathbf{M}_j \sim \mathcal{WN}(0, \mathbf{H}_j). \tag{2.34}$$

As stated, the goal of the Kalman-ILC is to achieve the best estimation of the next iteration error $\mathbf{E}_{j+1}$, and so of the next iteration disturbance $\mathbf{D}_{j+1}$, in order to decrease it by acting on the next iteration input vector $\mathbf{U}_{j+1}$. The estimated error based on the stochastic system (2.29) is equal to the expected error given all the past error measurements $\mathbf{E}^j$:

$$\hat{\mathbf{E}}_{j+1|j} := E[\,\mathbf{E}_{j+1}\,|\,\mathbf{E}^j\,] = E[\,\mathbf{F}\mathbf{V}_{j+1} + \mathbf{D}_{j+1} + \mathbf{M}_{j+1}\,|\,\mathbf{E}^j\,].$$

Since $E[\,\mathbf{M}_{j+1}\,|\,\mathbf{E}^j\,] = 0$, because the white noise vector $\mathbf{M}_j$ has zero-mean for every j, the one-step-ahead-predicted error as a function of the input update is then:

$$\begin{aligned}\hat{\mathbf{E}}_{j+1|j} &= E[\,\mathbf{F}\mathbf{V}_{j+1} + \mathbf{D}_{j+1}\,|\,\mathbf{E}^j\,]\\ &= E[\,\mathbf{F}\mathbf{V}_{j+1}\,|\,\mathbf{E}^j\,] + E[\,\mathbf{D}_{j+1}\,|\,\mathbf{E}^j\,]\\ &= \mathbf{F}\mathbf{V}_{j+1} + E[\,\mathbf{D}_{j+1}\,|\,\mathbf{E}^j\,].\end{aligned}$$

The K-ILC predicted tracking error is consequently:

$$\hat{\mathbf{E}}^{\text{K-ILC}}_{j+1}(\mathbf{U}_{j+1}) = \mathbf{F}\mathbf{V}_{j+1} + \hat{\mathbf{D}}_{j+1} \tag{2.35}$$

where the notation has been simplified: $\hat{\mathbf{E}}_{j+1} := \hat{\mathbf{E}}_{j+1|j}$ and $\hat{\mathbf{D}}_{j+1} := \hat{\mathbf{D}}_{j+1|j}$, in agreement with the notation previously discussed.

2.4.1 Algorithm

Specifically, the Kalman filter uses a recursive algorithm that exploits the knowledge of the present measurement to predict the *a priori* expected state vector and the expected error covariance of the state; in the next iteration, the new measurement are then used to correct the estimates. The filter works in a two-step process.

- In the iteration update step, the '*Predictor*' produces the *a priori* estimates of the current state variable and the variance matrix of the state prediction error:

$$\hat{\mathbf{D}}_{j+1|j} = \hat{\mathbf{D}}_{j|j} \tag{2.36}$$
$$\mathbf{P}_{j+1|j} = \mathbf{P}_{j|j} + \mathbf{\Sigma}_j, \tag{2.37}$$

where the state is the disturbance, and $\mathbf{P}_{j+1|j}$ is the variance matrix of the disturbance prediction error that is:

$$\mathbf{P}_{j+1|j} := E[\,\boldsymbol{\Gamma}_{j+1}\boldsymbol{\Gamma}_{j+1}^T\,], \tag{2.38}$$

in which the symbol $\boldsymbol{\Gamma}_{j+1}$ is used to define the state prediction error at the iteration $j+1$:

$$\begin{aligned}\boldsymbol{\Gamma}_{j+1} &= \mathbf{D}_{j+1} - E[\,\mathbf{D}_{j+1}\,|\,\mathbf{E}^j\,] \\ &= \mathbf{D}_{j+1} - \hat{\mathbf{D}}_{j+1|j}.\end{aligned} \tag{2.39}$$

- Once the outcome of the next measurement (j is shifted: $j \to j+1$) necessarily corrupted with noise is observed, the '*Corrector*' updates these estimates through the matrix gain $\mathbf{K}_j$:

  $$\mathbf{K}_j = \mathbf{P}_{j|j-1}(\mathbf{P}_{j|j-1} + \mathbf{H}_j)^{-1}. \tag{2.40}$$

  The Kalman gain is calculated in such a way that it minimize the *a posteriori* state error covariance. This gain weights the innovation, which contain the new information acquired from the measurement, to update the state vector, *i.e.* the disturbance, and also weights the state error covariance matrix for its update:

  $$\hat{\mathbf{D}}_{j|j} = \hat{\mathbf{D}}_{j|j-1} + \mathbf{K}_j(\mathbf{E}_j - \mathbf{F}\mathbf{V}_j - \hat{\mathbf{D}}_{j|j-1}) \tag{2.41}$$
  $$\mathbf{P}_{j|j} = (\mathbf{I} - \mathbf{K}_j)\mathbf{P}_{j|j-1}. \tag{2.42}$$

  The innovation $\mathbf{O}_j$ is equal to the term in parenthesis in equation (2.41) in fact:

  $$\begin{aligned}\mathbf{O}_j &:= \mathbf{E}_j - E[\,\mathbf{E}_j\,|\,\mathbf{E}^{j-1}\,] \\ &= \mathbf{E}_j - E[\,\mathbf{F}\mathbf{V}_j + \mathbf{D}_j + \mathbf{M}_j\,|\,\mathbf{E}^{j-1}\,] \\ &= \mathbf{E}_j - \mathbf{F}\mathbf{V}_j + \hat{\mathbf{D}}_{j|j-1},\end{aligned} \tag{2.43}$$

  since $E[\,\mathbf{M}_j\,] = 0$, as the noise $\mathbf{M}_j$ has zero-mean.

  The Kalman gain determines how heavily the measurement and the *a priori* estimate contribute to the computation of the state and, so, it determines how much the new measurement is reliable with respect to the prediction, based on the mathematical model (2.29).

  – If the measurement noise is small, the observation is trusted more and contributes to the state measurement update, *i.e.* to the calculation of $\mathbf{D}_{j|j}$, more than the *a priori* state estimate $\mathbf{D}_{j|j-1}$ does. In the case in which the measurement error covariance $\mathbf{H}_j$ is $\mathbf{H}_j \ll \mathbf{P}_{j|j-1}$, the Kalman gain approaches the identity matrix ($\mathbf{K}_j = \mathbf{I}$). Plugging the identity gain in (2.41), the *a posteriori* estimate of the disturbance becomes dependent on the current measurement and input 'shift' only:

  $$\hat{\mathbf{D}}_{j|j} = \mathbf{E}_j - \mathbf{F}\mathbf{V}_j. \tag{2.44}$$

– If, instead, the error in the *a priori* estimate is small, this *a priori* estimate is trusted more than measurement information, and the computation of $\mathbf{D}_{j|j}$ mostly comes from this estimate. Therefore, if the *a priori* error covariance matrix of the disturbance, $\mathbf{P}_{j|j-1}$, is close to zero: the the Kalman gain in the equation (2.40) goes to zero ($\mathbf{K}_j = 0$); this means that the contribution of the new measurement to the *a priori* disturbance is ignored:

$$\hat{\mathbf{D}}_{j|j} = \hat{\mathbf{D}}_{j|j-1}. \tag{2.45}$$

The algorithm is recursive; first a prediction is computed (iteration-update), then it is corrected based on the present iteration measurement (measurement-update). No additional past information is required.

To initialize the recursive algorithm, the initial disturbance and error covariance disturbance need to be assumed (for $j = 0$):

$$\hat{\mathbf{D}}_{0|-1} = \hat{\mathbf{D}}_0 \tag{2.46}$$

$$\mathbf{P}_{0|-1} = E[(\mathbf{D}_0 - \hat{\mathbf{D}}_{0|-1})(\mathbf{D}_0 - \hat{\mathbf{D}}_{0|-1})^T] = \mathbf{P}_0. \tag{2.47}$$

In fact, in absence of observations, no information is conveyed by the innovation and so the prediction is based on the *a priori* information only.

The Kalman estimator scheme is displayed in Figure 2.3.

Since the best one-step-ahead prediction of the disturbance is equal to the disturbance itself, the *a priori* prediction and *a posteriori* measurement update are easily combined:

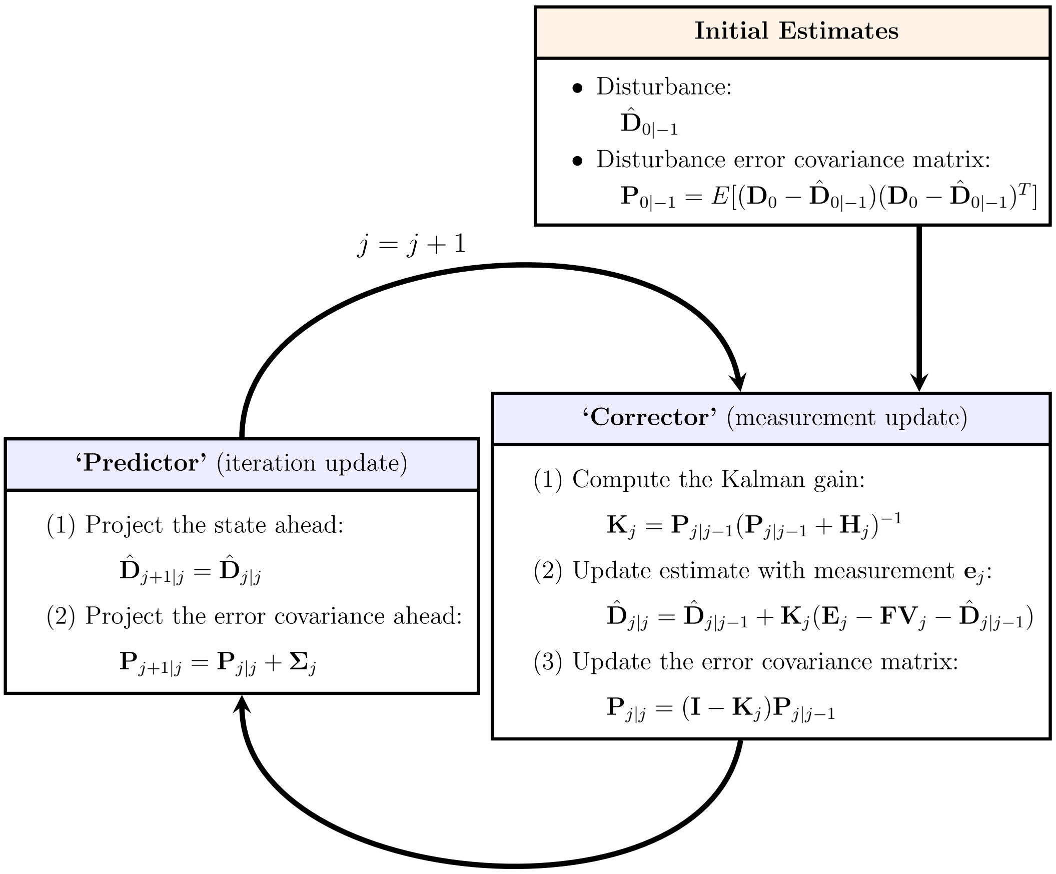
$$\begin{aligned}\hat{\mathbf{D}}_{j+1|j} &= \hat{\mathbf{D}}_{j|j}\\ &= \hat{\mathbf{D}}_{j|j-1} + \mathbf{K}_j(\mathbf{E}_j - \mathbf{F}\mathbf{V}_j - \hat{\mathbf{D}}_{j|j-1})\\ \mathbf{P}_{j+1|j} &= \mathbf{P}_{j|j} + \mathbf{\Sigma}_j\\ &= (\mathbf{I} - \mathbf{K}_j)\mathbf{P}_{j|j-1} + \mathbf{\Sigma}_j,\end{aligned}$$

with:

$$\mathbf{K}_j = \mathbf{P}_{j|j-1}(\mathbf{P}_{j|j-1} + \mathbf{H}_j)^{-1}.$$

The system of equations describing the disturbance estimator of the K-ILC thus is:

$$\begin{aligned}\mathbf{K}_j &= \mathbf{P}_{j|j-1}(\mathbf{P}_{j|j-1} + \mathbf{H}_j)^{-1}\\ \hat{\mathbf{D}}_{j+1|j} &= \hat{\mathbf{D}}_{j|j-1} + \mathbf{K}_j(\mathbf{E}_j - \mathbf{F}\mathbf{V}_j - \hat{\mathbf{D}}_{j|j-1})\\ \mathbf{P}_{j+1|j} &= (\mathbf{I} - \mathbf{K}_j)\mathbf{P}_{j|j-1} + \mathbf{\Sigma}_j.\end{aligned}$$



Initial Estimates
• Disturbance: $\hat{\mathbf{D}}_{0|-1}$
• Disturbance error covariance matrix: $\mathbf{P}_{0|-1} = E[(\mathbf{D}_0 - \hat{\mathbf{D}}_{0|-1})(\mathbf{D}_0 - \hat{\mathbf{D}}_{0|-1})^T]$
$j = j + 1$
'Corrector' (measurement update)
(1) Compute the Kalman gain: $\mathbf{K}_j = \mathbf{P}_{j|j-1}(\mathbf{P}_{j|j-1} + \mathbf{H}_j)^{-1}$
(2) Update estimate with measurement $\mathbf{e}_j$: $\hat{\mathbf{D}}_{j|j} = \hat{\mathbf{D}}_{j|j-1} + \mathbf{K}_j(\mathbf{E}_j - \mathbf{F}\mathbf{V}_j - \hat{\mathbf{D}}_{j|j-1})$
(3) Update the error covariance matrix: $\mathbf{P}_{j|j} = (\mathbf{I} - \mathbf{K}_j)\mathbf{P}_{j|j-1}$
'Predictor' (iteration update)
(1) Project the state ahead: $\hat{\mathbf{D}}_{j+1|j} = \hat{\mathbf{D}}_{j|j}$
(2) Project the error covariance ahead: $\mathbf{P}_{j+1|j} = \mathbf{P}_{j|j} + \mathbf{\Sigma}_j$

Figure 2.3: Kalman filtering

Re-writing the system with a simplified notation: $\mathbf{P}_j := \mathbf{P}_{j|j-1}$ and $\hat{\mathbf{D}}_j := \hat{\mathbf{D}}_{j|j-1}$, it yields:

$$\mathbf{K}_j = \mathbf{P}_j(\mathbf{P}_j + \mathbf{H}_j)^{-1} \tag{2.48}$$
$$\hat{\mathbf{D}}_{j+1} = \hat{\mathbf{D}}_j + \mathbf{K}_j(\mathbf{E}_j - \mathbf{F}\mathbf{V}_j - \hat{\mathbf{D}}_j) \tag{2.49}$$
$$\mathbf{P}_{j+1} = (\mathbf{I} - \mathbf{K}_j)\mathbf{P}_j + \mathbf{\Sigma}_j, \tag{2.50}$$

where the predictions of estimated initial disturbance $\hat{\mathbf{D}}_0$ and its error covariance matrix $\mathbf{P}_0$ are:

$$\hat{\mathbf{D}}_0 = \hat{\mathbf{D}}_{0|-1} \tag{2.51}$$
$$\mathbf{P}_0 = E[(\mathbf{D}_0 - \hat{\mathbf{D}}_0)(\mathbf{D}_0 - \hat{\mathbf{D}}_0)^T]. \tag{2.52}$$

For each iteration j, the measurement available of the error is used to correct the estimation of the disturbance and to update it so as to estimate the predicted

K-ILC tracking error:

$$\hat{\mathbf{E}}_{j+1}^{\text{K-ILC}}(\mathbf{U}_{j+1}) = \mathbf{F}\mathbf{V}_{j+1} + \hat{\mathbf{D}}_{j+1}. \tag{2.53}$$

Figure 2.4 shows the whole scheme that summaries the Kalman estimator of the ILC.

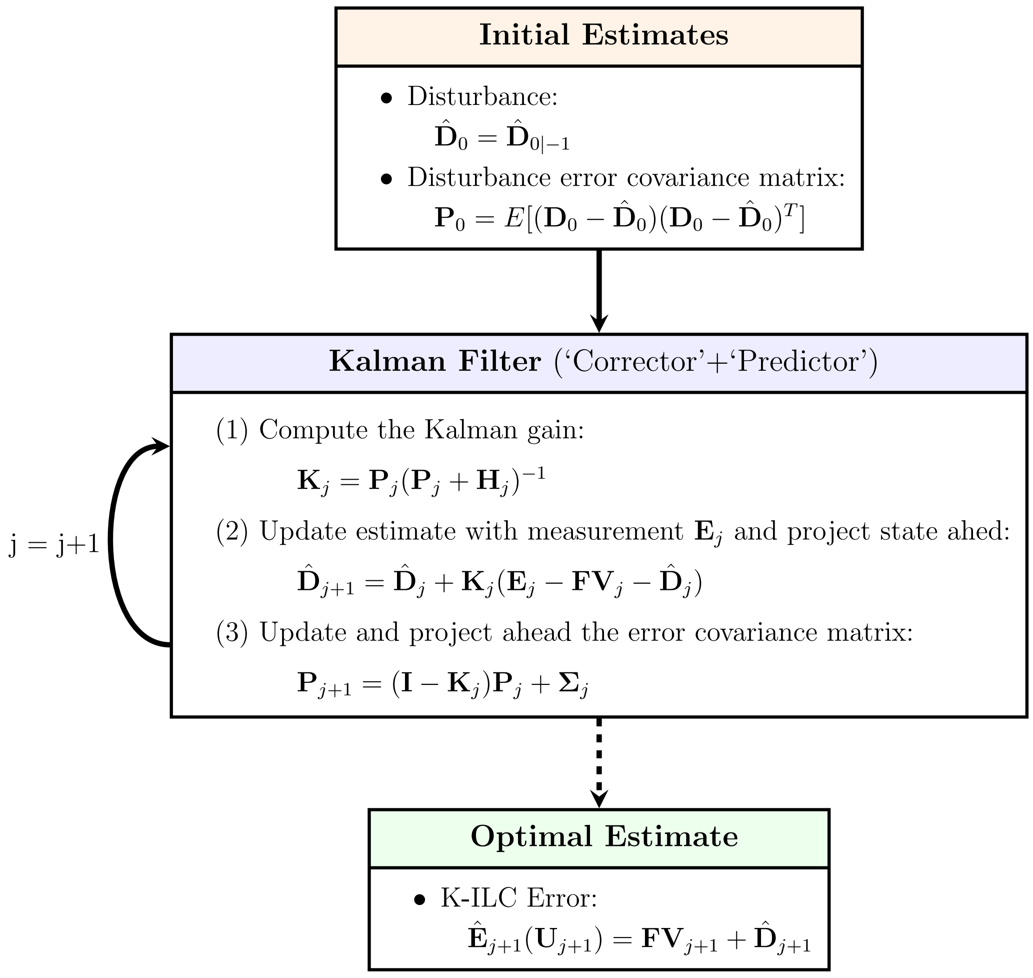

Initial Estimates
• Disturbance:
$\hat{\mathbf{D}}_0 = \hat{\mathbf{D}}_{0|-1}$
• Disturbance error covariance matrix:
$\mathbf{P}_0 = E[(\mathbf{D}_0 - \hat{\mathbf{D}}_0)(\mathbf{D}_0 - \hat{\mathbf{D}}_0)^T]$
Kalman Filter ('Corrector'+'Predictor')
(1) Compute the Kalman gain:
$\mathbf{K}_j = \mathbf{P}_j(\mathbf{P}_j + \mathbf{H}_j)^{-1}$
(2) Update estimate with measurement $\mathbf{E}_j$ and project state ahed:
$\hat{\mathbf{D}}_{j+1} = \hat{\mathbf{D}}_j + \mathbf{K}_j(\mathbf{E}_j - \mathbf{F}\mathbf{V}_j - \hat{\mathbf{D}}_j)$
(3) Update and project ahead the error covariance matrix:
$\mathbf{P}_{j+1} = (\mathbf{I} - \mathbf{K}_j)\mathbf{P}_j + \mathbf{\Sigma}_j$
j = j+1
Optimal Estimate
• K-ILC Error:
$\hat{\mathbf{E}}_{j+1}(\mathbf{U}_{j+1}) = \mathbf{F}\mathbf{V}_{j+1} + \hat{\mathbf{D}}_{j+1}$

Figure 2.4: Kalman Estimator

2.4.2 Kalman estimator - design parameters

The performance of the Kalman estimation step of the ILC can be improved by adjusting four different parameters.

- $\hat{\mathbf{D}}_0$ is the initial disturbance. It is typically excluded from the design parameters' choice since it is chosen to be zero as the nominal model is the best possible guess of the real model.

- $\mathbf{P}_0$ is the error covariance of the initial disturbance and it is defined as $\mathbf{P}_0 := E[(\mathbf{D}_0 - \hat{\mathbf{D}}_0)(\mathbf{D}_0 - \hat{\mathbf{D}}_0)^T]$. Given the expected initial disturbance null for hypothesis, it results in $\mathbf{P}_0 = E[\mathbf{D}_0\mathbf{D}_0^T]$. It is chosen to be a diagonal matrix with large positive elements to assure a rapid change in the disturbance estimation in the early learning trials. To further reduce the complexity of the problem the diagonal matrix proposed depends only on a single variable:

$$\mathbf{P}_0 = p_0\mathbf{I}_y, \quad p_0 > 0, \quad p_0 \in \mathbb{R},\ \mathbf{I}_y \in \mathbb{R}^{Nn_y \times Nn_y}. \tag{2.54}$$

- $\mathbf{\Sigma}_j$ is the covariance matrix that describes the null-mean Gaussian distribution of the white disturbance noise $\mathbf{\Omega}_j \sim \mathcal{WN}(0, \mathbf{\Sigma}_j)$. It captures the variations of the disturbance vector $\mathbf{D}$ during iterations. This matrix can also be modeled as a diagonal matrix depending on a common scalar term:

$$\mathbf{\Sigma}_j = \sigma_j\mathbf{I}_y, \quad \sigma_j > 0, \quad \sigma_j \in \mathbb{R},\ \mathbf{I}_y \in \mathbb{R}^{Nn_y \times Nn_y}. \tag{2.55}$$

  The scalar σ_j determines how the disturbance varies during the trials. Since $\mathbf{D}_j$ incorporates unmodeled dynamics, and, therefore, is input-dependent, σ is supposed to be larger at the beginning of learning, where the input $\mathbf{U}_j$ and, hence, $\mathbf{D}_j$ changes considerably, and smaller in the final steps of the learning process to avoid the disturbance to adapt to outliers and non-repetitive noise as the input converges. To simplify as much as possible the problem complexity, once again, the easiest choice is a constant value for σ_j: $\sigma_j = \bar{\sigma}$.

- $\mathbf{H}_j$ is the covariance matrix associated to the process and measurement white noise null-mean Gaussian distribution: $\mathbf{M}_j \sim \mathcal{WN}(0, \mathbf{H}_j)$. Based on the sensor noise characteristics, the covariance matrix is so modeled:

$$\mathbf{H}_j = \eta_j\mathbf{I}_y, \quad \eta_j > 0, \quad \eta_j \in \mathbb{R},\ \mathbf{I}_y \in \mathbb{R}^{Nn_y \times Nn_y}. \tag{2.56}$$

  Also in this case, the scalar term is supposed to be constant: $\eta_j = \bar{\eta}$. This simplification still holds in the case in which σ_j varies during iterations. This is due to the fact that there is a link between σ_j and η_j: the larger σ_j is with respect to η_j, the more the process model of the disturbance is considered reliable with respect the trustworthiness of the measurements (and vice versa). This is why, in the simplified case in which the two scalar parameters are iteration-invariant, it is possible to fix one of the two and to let the other fluctuate.

Additionally, scaling matrices $\mathbf{S}_E$ and $\mathbf{S}_H$ can be introduced when $\mathbf{D}_j$ and $\mathbf{Y}_j$ (and as a consequence $\mathbf{\Omega}_j$ and $\mathbf{M}_j$) represents different physical quantities with possible order of magnitudes differences:

$$\mathbf{\Sigma}_j = \mathbf{S}_E(\sigma_j\mathbf{I}_y)\mathbf{S}_E^T, \qquad \mathbf{H}_j = \mathbf{S}_H(\eta_j\mathbf{I}_y)\mathbf{S}_H^T. \tag{2.57}$$

In the model under study, this is not the case and, so, these scaling terms are not taken into account.

The parameters are displayed in Table 2.1.

Design Parameters	
• **Kalman**:	
Initial disturbance:	$\hat{\mathbf{D}}_0 = 0$
Initial covariance:	$\mathbf{P}_0 = p_0 \mathbf{I}_y, \quad p_0 \in \mathbb{R}_{>0}$
Dist. covariance:	$\mathbf{\Sigma}_j = \sigma_j \mathbf{I}_y, \quad \sigma_j \in \mathbb{R}_{>0}$
Error covariance:	$\mathbf{H}_j = \eta_j \mathbf{I}_y, \quad \eta_j \in \mathbb{R}_{>0}$
• **Weights**	
• **Constraints**	

Table 2.1: K-ILC design parameters - Kalman estimator

2.5 K-ILC input update

The Kalman-ILC is completed with the learning update step. Given the predicted error $\hat{\mathbf{E}}_{j+1}$ provided by the (Kalman) estimator, the objective of the updated step is to find the optimal next iteration input $\mathbf{U}_{j+1}$ (with $\mathbf{U}_{j+1} \in \mathbb{R}^{Nn_u}$) that reduces the next iteration error by compensating for the disturbance $\hat{\mathbf{D}}_{j+1}$.

Specifically, the next iteration input vector is computed minimizing a cost function that weights the predicted error as well as the input changes (which happen both during the trial and from an iteration to the next).

The input update optimization problem is formulated as follows:

$$\mathbf{U}_{j+1} = \underset{\mathbf{U'}_{j+1}}{\operatorname{argmin}} \left\{ \tilde{J}_{j+1}(\mathbf{U}'_{j+1}) \right\}, \tag{2.58}$$

subject to the following 'hard' constraints :

$$\begin{cases} \mathbf{Z}\mathbf{U}_{j+1} \leq \mathbf{Q}_{max} \\ \mathbf{Z}_{eq}\mathbf{U}_{j+1} = \mathbf{Q}_{eq}. \end{cases} \tag{2.59}$$

$\tilde{J}_{j+1}$ is the cost function dependent on the updated input $\mathbf{U}_{j+1}$, whereas $\mathbf{Z}$, $\mathbf{Q}_{max}$, $\mathbf{Z}_{eq}$ and $\mathbf{Q}_{eq}$ defines the inequality (the first two terms) and equality (the last two) constraints.

The cost function is presented in subsection 2.5.1, in subsection 2.5.3, instead, the input constraints are explicitly defined. The optimization scheme is shown in Figure 2.5.

2.5.1 Cost function

The optimal input update is stated in its most general form as the minimization of the following cost function:

$$\begin{aligned} J_{j+1}(\mathbf{u}_{j+1}) &= \|\mathbf{W}_e\hat{\mathbf{E}}_{j+1}\|_p^a + \|\mathbf{W}_{\Delta u}\mathbf{\Delta U}_{j+1}\|_p^a \\ &\quad + \|\mathbf{W}_u\mathbf{U}_{j+1}\|_p^a + \|\mathbf{W}_v\mathbf{V}_{j+1}\|_p^a, \end{aligned} \tag{2.60}$$

where $\|\cdot\|_p^a$ is the p-norm raised to the power of a, $\mathbf{W}_e$, $\mathbf{W}_{\Delta U}$, $\mathbf{W}_u$ and $\mathbf{W}_v$ are weighting matrices acting respectively on the estimated next trial error $\hat{\mathbf{E}}_{j+1}$, the input change $\mathbf{\Delta U}_{j+1} = \mathbf{U}_{j+1} - \mathbf{U}_j$, the next iteration input $\mathbf{U}_{j+1}$ and the next 'shifted' input $\mathbf{V}_{j+1} = \mathbf{U}_{j+1} - \mathbf{U}_{nom}$.

The predicted error $\hat{\mathbf{E}}_{j+1}$ is explicitly defined as:

$$\hat{\mathbf{E}}_{j+1} := \hat{\mathbf{E}}_{j+1|j} = E[\mathbf{E}_{j+1}|\mathbf{Y}_0, \mathbf{Y}_1, \ldots, \mathbf{Y}_j] = \mathbf{F}\mathbf{V}_{j+1} + \hat{\mathbf{D}}_{j+1}. \tag{2.61}$$

Outlining the explicit error, the cost function becomes:

$$\begin{aligned} J_{j+1}^{(K-ILC)}(\mathbf{U}_{j+1}) &= \|\mathbf{W}_e(\mathbf{F}\mathbf{V}_{j+1} + \hat{\mathbf{D}}_{j+1})\|_p^a + \|\mathbf{W}_{\Delta u}\mathbf{\Delta U}_{j+1}\|_p^a \\ &\quad + \|\mathbf{W}_u\mathbf{U}_{j+1}\|_p^a + \|\mathbf{W}_v\mathbf{V}_{j+1}\|_p^a. \end{aligned} \tag{2.62}$$

The optimization problem can be done numerically considering any p-norm to the power of a. The optimization problem considered is a standard quadratic convex problem with $p = 2$ and $a = 2$ (squared 2-norm). The Euclidean norm is used because it minimizes the sum of the deviations of the output along the desired trajectory, focusing the weighting cost on large errors. Moreover, this type of optimization has the advantage that, if there exists a local minimum, and so, if an input compliant with the constraints is feasible, this local minimum is globally optimal. On top of that, quadratic optimization problem can be solved faster and more efficiently using **MATLAB**® standard functions or software packages such as the **IMB CPLEX® Optimizer**. For $p = 2$ and $a = 2$, the cost function can be redefined explicitly in terms of the input update. The cost function term linked to the error is then:

$$\begin{aligned} J_{j+1}^{(E)}(\mathbf{v}_{j+1}) &= [\mathbf{W}_e(\mathbf{F}\mathbf{V}_{j+1} + \hat{\mathbf{D}}_{j+1})]^T[\mathbf{W}_e(\mathbf{F}\mathbf{V}_{j+1} + \hat{\mathbf{D}}_{j+1})] \\ &= \mathbf{V}_{j+1}^T\mathbf{F}^T\mathbf{W}_e^T\mathbf{W}_e\mathbf{F}\mathbf{V}_{j+1} + \mathbf{V}_{j+1}^T\mathbf{F}^T\mathbf{W}_e^T\mathbf{W}_e\hat{\mathbf{D}}_{j+1} \\ &\quad + \hat{\mathbf{D}}_{j+1}^T\mathbf{W}_e^T\mathbf{W}_e\mathbf{F}\mathbf{V}_{j+1} + \hat{\mathbf{D}}_{j+1}^T\mathbf{W}_e^T\mathbf{W}_e\hat{\mathbf{D}}_{j+1} \\ &= \mathbf{V}_{j+1}^T\mathbf{F}^T\tilde{\mathbf{W}}_e\mathbf{F}\mathbf{V}_{j+1} + 2\hat{\mathbf{D}}_{j+1}^T\tilde{\mathbf{W}}_e\mathbf{F}\mathbf{V}_{j+1} + \hat{\mathbf{D}}_{j+1}^T\tilde{\mathbf{W}}_e\hat{\mathbf{D}}_{j+1}, \end{aligned}$$

with: $\tilde{\mathbf{W}}_e := \mathbf{W}_e^T\mathbf{W}_e$.

Defining the error cost as a function of the input update it yields:

$$\begin{aligned} J_{j+1}^{(E)}(\mathbf{U}_{j+1}) &= (\mathbf{U}_{j+1} - \mathbf{U}_{nom})^T\mathbf{F}^T\tilde{\mathbf{W}}_e\mathbf{F}(\mathbf{U}_{j+1} - \mathbf{U}_{nom}) \\ &\quad + 2\hat{\mathbf{D}}_{j+1}^T\tilde{\mathbf{W}}_e\mathbf{F}(\mathbf{U}_{j+1} - \mathbf{U}_{nom}) + \mathbf{D}_{j+1}^T\tilde{\mathbf{W}}_e\mathbf{D}_{j+1} \\ &= \mathbf{U}_{j+1}^T\mathbf{F}^T\tilde{\mathbf{W}}_e\mathbf{F}\mathbf{U}_{j+1} - 2\mathbf{U}_{nom}^T\mathbf{F}^T\tilde{\mathbf{W}}_e\mathbf{F}\mathbf{U}_{j+1} + \mathbf{U}_{nom}^T\mathbf{F}^T\tilde{\mathbf{W}}_e\mathbf{F}\mathbf{U}_{nom} \\ &\quad + 2\hat{\mathbf{D}}_{j+1}^T\tilde{\mathbf{W}}_e\mathbf{F}\mathbf{U}_{j+1} - 2\hat{\mathbf{D}}_{j+1}^T\tilde{\mathbf{W}}_e\mathbf{F}\mathbf{U}_{nom} + \mathbf{D}_{j+1}^T\tilde{\mathbf{W}}_e\mathbf{D}_{j+1}. \end{aligned}$$

The change of input cost term as a function of $\mathbf{U}_{j+1}$ is:

$$\begin{aligned}J^{(\Delta U)}_{j+1}(\mathbf{U}_{j+1}) &= \mathbf{\Delta U}^T_{j+1}\mathbf{W}^T_{\Delta u}\mathbf{W}_{\Delta u}\mathbf{\Delta U}_{j+1}\\ &= \mathbf{\Delta U}^T_{j+1}\tilde{\mathbf{W}}_{\Delta u}\mathbf{\Delta U}_{j+1}\\ &= (\mathbf{U}_{j+1}-\mathbf{U}_j)^T\tilde{\mathbf{W}}_{\Delta u}(\mathbf{U}_{j+1}-\mathbf{U}_j)\\ &= \mathbf{U}^T_{j+1}\tilde{\mathbf{W}}_{\Delta u}\mathbf{U}_{j+1} - 2\mathbf{U}^T_j\tilde{\mathbf{W}}_{\Delta u}\mathbf{U}_{j+1} + \mathbf{U}^T_j\tilde{\mathbf{W}}_{\Delta u}\mathbf{U}_j,\end{aligned}$$

with: $\tilde{\mathbf{W}}_{\Delta u} := \mathbf{W}^T_{\Delta u}\mathbf{W}_{\Delta u}$.
The cost function weight referred to the input update is instead:

$$\begin{aligned}J^{(U)}_{j+1}(\mathbf{U}_{j+1}) &= \mathbf{U}^T_{j+1}\mathbf{W}^T_u\mathbf{W}_u\mathbf{U}_{j+1}\\ &= \mathbf{U}^T_{j+1}\tilde{\mathbf{W}}_u\mathbf{U}_{j+1},\end{aligned}$$

with: $\tilde{\mathbf{W}}_u := \mathbf{W}^T_u\mathbf{W}_u$.
Lastly, the 'shifted' input cost is equal to:

$$\begin{aligned}J^{(V)}_{j+1}(\mathbf{U}_{j+1}) &= \mathbf{V}^T_{j+1}\mathbf{W}^T_v\mathbf{W}_v\mathbf{V}_{j+1}\\ &= (\mathbf{U}_{j+1}-\mathbf{U}_{nom})^T\tilde{\mathbf{W}}_u(\mathbf{U}_{j+1}-\mathbf{U}_{nom})\\ &= \mathbf{U}^T_{j+1}\tilde{\mathbf{W}}_u\mathbf{U}_{j+1} - 2\mathbf{U}^T_{nom}\tilde{\mathbf{W}}_u\mathbf{U}_{j+1} + \mathbf{U}^T_{nom}\tilde{\mathbf{W}}_u\mathbf{U}_{nom},\end{aligned}$$

with: $\tilde{\mathbf{W}}_v := \mathbf{W}^T_v\mathbf{W}_v$.
All the cost function terms, dependent on the input update only[7], are displayed below:

$$\begin{aligned}\tilde{J}^{(E)}_{j+1} &= \frac{1}{2}\left(\mathbf{D}^T_{j+1}(2\mathbf{F}^T\tilde{\mathbf{W}}_e\mathbf{F})\mathbf{U}_{j+1}\right) + (2\hat{\mathbf{D}}^T_{j+1}\tilde{\mathbf{W}}_e\mathbf{F} - 2\mathbf{U}^T_{nom}\mathbf{F}^T\tilde{\mathbf{W}}_e\mathbf{F})\mathbf{U}_{j+1}\\ \tilde{J}^{(\Delta u)}_{j+1} &= \frac{1}{2}\left(\mathbf{U}^T_{j+1}(2\tilde{\mathbf{W}}_{\Delta u})\mathbf{U}_{j+1}\right) + (-2\mathbf{U}^T_j\tilde{\mathbf{W}}_{\Delta u})\mathbf{U}_{j+1}\\ \tilde{J}^{(U)}_{j+1} &= \frac{1}{2}\left(\mathbf{U}^T_{j+1}(2\tilde{\mathbf{W}}_u)\mathbf{U}_{j+1}\right)\\ \tilde{J}^{(V)}_{j+1} &= \frac{1}{2}\left(\mathbf{U}^T_{j+1}(2\tilde{\mathbf{W}}_v)\mathbf{U}_{j+1}\right) + (-2\mathbf{U}^T_{nom}\tilde{\mathbf{W}}_v)\mathbf{U}_{j+1}.\end{aligned}$$

Combining all previous terms, it yields:

$$\begin{aligned}\tilde{J}_{j+1}(\mathbf{U}_{j+1}) = \frac{1}{2}&\left(\mathbf{U}^T_{j+1}(2\mathbf{F}^T\tilde{\mathbf{W}}_e\mathbf{F} + 2\tilde{\mathbf{W}}_{\Delta u} + 2\tilde{\mathbf{W}}_u + 2\tilde{\mathbf{W}}_v)\mathbf{U}_{j+1}\right)\\ &+ (2\hat{\mathbf{D}}^T_{j+1}\tilde{\mathbf{W}}_e\mathbf{F} - 2\mathbf{U}^T_{nom}\mathbf{F}^T\tilde{\mathbf{W}}_e\mathbf{F} - 2\mathbf{U}^T_j\tilde{\mathbf{W}}_{\Delta u} - 2\mathbf{U}^T_{nom}\tilde{\mathbf{W}}_v)\mathbf{U}_{j+1}.\end{aligned}$$

The cost function of the optimization problem written in short is ultimately:

$$\tilde{J}_{j+1}(\mathbf{U}_{j+1}) = \frac{1}{2}\mathbf{U}^T_{j+1}\tilde{\mathbf{H}}\mathbf{U}_{j+1} + \tilde{\mathbf{f}}\mathbf{U}_{j+1}, \tag{2.63}$$

[7]The tilde notation is used to point out the deletion of constant contributions.

with:

$$\tilde{\mathbf{H}} := 2(\mathbf{F}^T\tilde{\mathbf{W}}_e\mathbf{F} + \tilde{\mathbf{W}}_{\Delta u} + \tilde{\mathbf{W}}_u + \tilde{\mathbf{W}}_v), \tag{2.64}$$

$$\tilde{\mathbf{f}} := 2\hat{\mathbf{D}}_{j+1}^T\tilde{\mathbf{W}}_e\mathbf{F} - 2\mathbf{U}_{nom}^T\mathbf{F}^T\tilde{\mathbf{W}}_e\mathbf{F} - 2\mathbf{U}_j^T\tilde{\mathbf{W}}_{\Delta u} - 2\mathbf{U}_{nom}^T\tilde{\mathbf{W}}_v. \tag{2.65}$$

In summary, the input update vector is computed solving the optimization problem.

$$\mathbf{U}_{j+1} = \underset{\mathbf{U'}_{j+1}}{\operatorname{argmin}} \left\{ \frac{1}{2}\mathbf{U'}_{j+1}^T\tilde{\mathbf{H}}\mathbf{U'}_{j+1} + \tilde{\mathbf{f}}\mathbf{U'}_{j+1} \right\} \tag{2.66}$$

subject to inequality and equality input constraints.

2.5.2 Weights - design parameters

Four different weight design parameters allow an adaptation of the optimization problem (not accounting for the 'hard constraints' which are defined in the next section).

- $\mathbf{W}_e$ is the matrix defined in $\mathbb{R}^{Nn_y \times Nn_y}$ that weights and scales the estimated one-step-ahead error signal. The matrix is designed to balance the magnitude of possible different physical quantities for MIMO system. It is also useful for stressing the importance in accuracy of specific states, augmenting the cost function associated with these states. Since the matrix weights the error vector defined in lifted-domain, the weight can additionally be adjusted to give importance to defined time-frames, *i.e.*, specific parts of the trajectory (*e.g.* initial and final phases).

  To distinguish the weighting actions, the matrix can be decomposed in three diagonal matrices, one to equalize the range of magnitude, another to focus the emphasis on specific states and another one to change the weighting along the trajectory. In the present dissertation, because of the nature of the problem (with homogeneous physical quantities) and in order to simplify the process, the chosen matrix is, once again, diagonal with equal coefficients:

  $$\mathbf{W}_e = w_e\mathbf{I}_y, \quad w_e > 0, \quad w_e \in \mathbb{R},\ \mathbf{I}_y \in \mathbb{R}^{Nn_y \times Nn_y}. \tag{2.67}$$

- $\mathbf{W}_{\Delta u}$ is the matrix penalizing the change of input $\Delta\mathbf{u}_{j+1}$ from one iteration to the next. Once more, the weight chosen is a diagonal matrix:

  $$\mathbf{W}_{\Delta u} = w_{\Delta u}\mathbf{I}_u, \quad w_{\Delta u} > 0, \quad w_{\Delta u} \in \mathbb{R},\ \mathbf{I}_u \in \mathbb{R}^{Nn_u \times Nn_u}. \tag{2.68}$$

  It is convenient if the designer wants to enforce gradual smoothness of the ILC update input from one iteration to the next. The scalar weight $w_{\Delta u}$ is closely related to the weight w_e. The more emphasis is given to the tracking error w_e with respect to $w_{\Delta u}$ in terms of magnitude, the faster the output vector will converge to the desired one, penalizing, on the other side, the gradualness and smoothness of learning. Because of that, a higher value of $w_{\Delta u}$ may worsen the learning performance.

Design Parameters		
• **Kalman**:	$\hat{\mathbf{D}}_0,\ \mathbf{P}_0, \mathbf{\Sigma}_j, \mathbf{H}_j$	
• **Weights**:		
Error	$\mathbf{W}_e = w_e \mathbf{I}_y,$	$w_e \in \mathbb{R}$
Iter. inputs diff.	$\mathbf{W}_{\Delta u} = w_{\Delta u} \mathbf{1}_u,$	$w_{\Delta u} \in \mathbb{R}$
Input	$\mathbf{W}_u = 0$	
Nom. inputs diff.	$\mathbf{W}_v = 0$	
• **Constraints**		

Table 2.2: K-ILC design parameters - weights

- $\mathbf{W}_u$ is the matrix acting in minimizing the input and, therefore, the converged solution $\mathbf{U}_\infty$. It represent a 'soft' constraint and does not affect the ILC robustness and convergence properties. The weight is omitted from the current analysis since the limits in terms of absolute value (and derivatives) of the input, are explicitly accounted in the inequality constraints.

- $\mathbf{W}_v$ is the weight matrix which reduces the asymptotic vector $\mathbf{V}_\infty$, which represents the difference between the actual input and the nominal input at convergence. The same treatment of $\mathbf{W}_u$ applies for $\mathbf{W}_v$.

  These weight parameters are summarized in Table 2.2.

2.5.3 Constraints - design parameters

The designed optimization problem includes 'hard' constraints express in terms of inequality and equality constraints.

Inequality constraints

First the definition of the input vector $\mathbf{u}_j$ is recalled:

$$\mathbf{U}_j = \begin{bmatrix} \tilde{\mathbf{u}}_j(0)^T & \tilde{\mathbf{u}}_j(1)^T & \cdots & \tilde{\mathbf{u}}_j(N-1)^T \end{bmatrix}^T, \qquad \mathbf{U}_j \in \mathbb{R}^{Nn_u},$$

where the vector $\tilde{\mathbf{u}}_j(k) \in \mathbb{R}^{n_u}$ in this section represents the inputs of the discrete dynamic system obtained by sampling the continuous signals $\tilde{\mathbf{u}}_j(t)$.

The input inequality constraints imposed are expressed in terms of maximum and minimum absolute values and in terms of first and second order derivatives:

$$\tilde{\mathbf{u}}_{min} \leq \tilde{\mathbf{u}}_j(t) \leq \tilde{\mathbf{u}}_{max} \tag{2.69}$$

$$\left(\frac{d\tilde{\mathbf{u}}}{dt}\right)_{min} \leq \frac{d\tilde{\mathbf{u}}_j}{dt} \leq \left(\frac{d\tilde{\mathbf{u}}}{dt}\right)_{max} \tag{2.70}$$

$$\left(\frac{d^2\tilde{\mathbf{u}}}{dt^2}\right)_{min} \leq \frac{d^2\tilde{\mathbf{u}}_j}{dt^2} \leq \left(\frac{d^2\tilde{\mathbf{u}}}{dt^2}\right)_{max}. \tag{2.71}$$

The relations referred to the continuous time variable t are analogous to the one actually used, related to the discrete variable k:

$$\tilde{\mathbf{u}}_{min} \leq \tilde{\mathbf{u}}_j(k) \leq \tilde{\mathbf{u}}_{max} \tag{2.72}$$

$$\left(\frac{d\tilde{\mathbf{u}}}{dk}\right)_{min} \leq \frac{\Delta\tilde{\mathbf{u}}_j}{\Delta k} \leq \left(\frac{d\tilde{\mathbf{u}}}{dk}\right)_{max} \tag{2.73}$$

$$\left(\frac{d^2\tilde{\mathbf{u}}}{dk^2}\right)_{min} \leq \frac{\Delta^2\tilde{\mathbf{u}}_j}{\Delta k^2} \leq \left(\frac{d^2\tilde{\mathbf{u}}}{dk^2}\right)_{max}, \tag{2.74}$$

where the following approximations hold:

$$\left(\frac{\Delta\tilde{\mathbf{u}}}{\Delta k}\right)_{max/min} \approx \left(\frac{d\tilde{\mathbf{u}}}{dt}\right)_{max/min}$$

$$\left(\frac{\Delta^2\tilde{\mathbf{u}}}{\Delta k^2}\right)_{max/min} \approx \left(\frac{d^2\tilde{\mathbf{u}}}{d^2 t}\right)_{max/min}.$$

The first and second discrete derivatives are defined according to the following approximations:

- **1st** discrete derivative:
  - **Forward** approximation:

$$\left.\frac{d\tilde{\mathbf{u}}_j}{dt}\right|_{\bar{k}} \approx \left.\frac{\Delta\tilde{\mathbf{u}}_j}{\Delta k}^{(+)}\right|_{\bar{k}} = \frac{\tilde{\mathbf{u}}_j(\bar{k}+1) - \tilde{\mathbf{u}}_j(\bar{k})}{t_s}; \tag{2.75}$$

  - **Centered** approximation:

$$\left.\frac{d\tilde{\mathbf{u}}_j}{dt}\right|_{\bar{k}} \approx \left.\frac{\Delta\tilde{\mathbf{u}}_j}{\Delta k}^{(c)}\right|_{\bar{k}} = \frac{\tilde{\mathbf{u}}_j(\bar{k}+1) - \tilde{\mathbf{u}}_j(\bar{k}-1)}{t_s}; \tag{2.76}$$

  - **Backward** approximation:

$$\left.\frac{d\tilde{\mathbf{u}}_j}{dt}\right|_{\bar{k}} \approx \left.\frac{\Delta\tilde{\mathbf{u}}_j}{\Delta k}^{(-)}\right|_{\bar{k}} = \frac{\tilde{\mathbf{u}}_j(\bar{k}) - \tilde{\mathbf{u}}_j(\bar{k}-1)}{t_s}; \tag{2.77}$$

- **2nd** discrete derivative:

  - **Forward** approximation:

$$\left.\frac{d^2\tilde{\mathbf{u}}_j}{dt^2}\right|_{\bar{k}} \approx \left.\frac{\Delta^2\tilde{\mathbf{u}}_j}{\Delta k^2}^{(+)}\right|_{\bar{k}} = \frac{\tilde{\mathbf{u}}_j(\bar{k}+2) - 2\tilde{\mathbf{u}}_j(\bar{k}+1) + \tilde{\mathbf{u}}_j(\bar{k})}{t_s^2}; \tag{2.78}$$

  - **Centered** approximation:

$$\left.\frac{d^2\tilde{\mathbf{u}}_j}{dt^2}\right|_{\bar{k}} \approx \left.\frac{\Delta^2\tilde{\mathbf{u}}_j}{\Delta k^2}^{(c)}\right|_{\bar{k}} = \frac{\tilde{\mathbf{u}}_j(\bar{k}+1) - 2\tilde{\mathbf{u}}_j(\bar{k}) + \tilde{\mathbf{u}}_j(\bar{k}-1)}{t_s^2}; \tag{2.79}$$

  - **Backward** approximation:

$$\left.\frac{d^2\tilde{\mathbf{u}}_j}{dt^2}\right|_{\bar{k}} \approx \left.\frac{\Delta^2\tilde{\mathbf{u}}_j}{\Delta k^2}^{(-)}\right|_{\bar{k}} = \frac{\tilde{\mathbf{u}}_j(\bar{k}) - 2\tilde{\mathbf{u}}_j(\bar{k}-1) + \tilde{\mathbf{u}}_j(\bar{k}-2)}{t_s^2}. \tag{2.80}$$

The inequality constraints (equations (2.72), (2.73) and (2.74)) are written in the lifted-domain for the discrete time steps k (with $k \in \mathcal{K}$). The forward approximation is used to express in matrix form the derivatives up to $k = N - 3$. The discrete derivatives of the last two time steps are computed using a backward approximation.

The computation of the inequality constraint is reported below.

$$
\begin{aligned}
\tilde{\mathbf{u}}_j(0) &\leq \tilde{\mathbf{u}}_{max} \\
\tilde{\mathbf{u}}_j(0) &\geq -\tilde{\mathbf{u}}_{min} \\
\left.\frac{\Delta \tilde{\mathbf{u}}_j}{\Delta t}^{(+)}\right|_{(0)} = \frac{\tilde{\mathbf{u}}_j(1) - \tilde{\mathbf{u}}_j(0)}{t_s} &\leq \left(\frac{d\tilde{\mathbf{u}}}{dt}\right)_{max} \\
\left.\frac{\Delta \tilde{\mathbf{u}}_j}{\Delta t}^{(+)}\right|_{(0)} = \frac{\tilde{\mathbf{u}}_j(1) - \tilde{\mathbf{u}}_j(0)}{t_s} &\geq -\left(\frac{d\tilde{\mathbf{u}}}{dt}\right)_{min} \\
\left.\frac{\Delta^2 \tilde{\mathbf{u}}_j}{\Delta t^2}^{(+)}\right|_{(0)} = \frac{\tilde{\mathbf{u}}_j(2) - 2\tilde{\mathbf{u}}_j(1) + \tilde{\mathbf{u}}_j(0)}{t_s^2} &\leq \left(\frac{d^2\tilde{\mathbf{u}}}{dt^2}\right)_{max} \\
\left.\frac{\Delta \tilde{\mathbf{u}}_j}{\Delta t}^{(+)}\right|_{(0)} = \frac{\tilde{\mathbf{u}}_j(2) - 2\tilde{\mathbf{u}}_j(1) + \tilde{\mathbf{u}}_j(0)}{t_s^2} &\geq -\left(\frac{d^2\tilde{\mathbf{u}}}{dt^2}\right)_{min} \\
&\vdots \\
\tilde{\mathbf{u}}_j(N{-}1) &\leq \tilde{\mathbf{u}}_{max} \\
\tilde{\mathbf{u}}_j(N{-}1) &\geq -\tilde{\mathbf{u}}_{min} \\
\left.\frac{\Delta \tilde{\mathbf{u}}_j}{\Delta t}^{(-)}\right|_{(N-1)} = \frac{\tilde{\mathbf{u}}_j(N{-}1) - \tilde{\mathbf{u}}_j(N{-}2)}{t_s} &\leq \left(\frac{d\tilde{\mathbf{u}}}{dt}\right)_{max} \\
\left.\frac{\Delta \tilde{\mathbf{u}}_j}{\Delta t}^{(-)}\right|_{(N-1)} = \frac{\tilde{\mathbf{u}}_j(N{-}1) - \tilde{\mathbf{u}}_j(N{-}2)}{t_s} &\geq -\left(\frac{d\tilde{\mathbf{u}}}{dt}\right)_{min} \\
\left.\frac{\Delta^2 \tilde{\mathbf{u}}_j}{\Delta t^2}^{(-)}\right|_{(N-1)} = \frac{\tilde{\mathbf{u}}_j(N{-}1) - 2\tilde{\mathbf{u}}_j(N{-}2) + \tilde{\mathbf{u}}_j(N{-}3)}{t_s^2} &\leq \left(\frac{d^2\tilde{\mathbf{u}}}{dt^2}\right)_{max} \\
\left.\frac{\Delta^2 \tilde{\mathbf{u}}_j}{\Delta t^2}^{(-)}\right|_{(N-1)} = \frac{\tilde{\mathbf{u}}_j(N{-}1) - 2\tilde{\mathbf{u}}_j(N{-}2) + \tilde{\mathbf{u}}_j(N{-}3)}{t_s^2} &\geq -\left(\frac{d^2\tilde{\mathbf{u}}}{dt^2}\right)_{min}
\end{aligned}
$$

Rewriting the above equations in the following way:

$$\begin{aligned}
\tilde{\mathbf{u}}_j(0) &\leq \tilde{\mathbf{u}}_{max} \\
-\tilde{\mathbf{u}}_j(0) &\leq \tilde{\mathbf{u}}_{min} \\
-\tilde{\mathbf{u}}_j(0) + \tilde{\mathbf{u}}_j(1) &\leq t_s \left(\frac{d\tilde{\mathbf{u}}}{dt}\right)_{max} \\
\tilde{\mathbf{u}}_j(0) - \tilde{\mathbf{u}}_j(1) &\leq t_s \left(\frac{d\tilde{\mathbf{u}}}{dt}\right)_{min} \\
\tilde{\mathbf{u}}_j(0) - 2\tilde{\mathbf{u}}_j(1) + \tilde{\mathbf{u}}_j(2) &\leq t_s^2 \left(\frac{d^2\tilde{\mathbf{u}}}{dt^2}\right)_{max} \\
-\tilde{\mathbf{u}}_j(0) + 2\tilde{\mathbf{u}}_j(1) - \tilde{\mathbf{u}}_j(2) &\leq t_s^2 \left(\frac{d^2\tilde{\mathbf{u}}}{dt^2}\right)_{min} \\
&\vdots \\
\tilde{\mathbf{u}}_j(N{-}1) &\leq \tilde{\mathbf{u}}_{max} \\
-\tilde{\mathbf{u}}_j(N{-}1) &\leq \tilde{\mathbf{u}}_{min} \\
-\tilde{\mathbf{u}}_j(N{-}2) + \tilde{\mathbf{u}}_j(N{-}1) &\leq t_s \left(\frac{d\tilde{\mathbf{u}}}{dt}\right)_{max} \\
\tilde{\mathbf{u}}_j(N{-}2) - \tilde{\mathbf{u}}_j(N{-}1) &\leq t_s \left(\frac{d\tilde{\mathbf{u}}}{dt}\right)_{min} \\
\tilde{\mathbf{u}}_j(N{-}3) - 2\tilde{\mathbf{u}}_j(N{-}2) + \tilde{\mathbf{u}}_j(N{-}1) &\leq t_s^2 \left(\frac{d^2\tilde{\mathbf{u}}}{dt^2}\right)_{max} \\
-\tilde{\mathbf{u}}_j(N{-}3) + 2\tilde{\mathbf{u}}_j(N{-}2) - \tilde{\mathbf{u}}_j(N{-}1) &\leq t_s^2 \left(\frac{d^2\tilde{\mathbf{u}}}{dt^2}\right)_{min},
\end{aligned}$$

it is possible to defines the inequality constraints in a compact form:

$$\mathbf{Z}\mathbf{U}_j \leq \mathbf{Q}_{max}. \tag{2.81}$$

Where:

$$\mathbf{Z} = \begin{bmatrix} \mathbf{Z}_0 & \mathbf{0}_{s\times 1} & \cdots & \cdots & \mathbf{0}_{s\times 1} \\ \mathbf{0}_{s\times 1} & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \mathbf{Z}_p & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \mathbf{0}_{s\times 1} \\ \mathbf{0}_{s\times 1} & \cdots & \mathbf{0}_{s\times 1} & \mathbf{0}_{s\times 1} & \mathbf{Z}_{N-3} \\ \mathbf{0}_{s\times 1} & \cdots & \mathbf{0}_{s\times 1} & \mathbf{z}_{N-2} & \mathbf{Z}_{N-2} \\ \mathbf{0}_{s\times 1} & \cdots & \mathbf{0}_{s\times 1} & \mathbf{0}_{s\times 1} & \mathbf{Z}_{N-1} \end{bmatrix}, \qquad \mathbf{Q}_{max} = \begin{bmatrix} \mathbf{q}_{max_0} \\ \vdots \\ \mathbf{q}_{max_p} \\ \vdots \\ \mathbf{q}_{max_{N-3}} \\ \mathbf{q}_{max_{N-2}} \\ \mathbf{q}_{max_{N-1}} \end{bmatrix},$$

with: $\mathbf{Z} \in \mathbb{R}^{6n_uN\times n_uN}$, $s = 6n_u$, $\mathbf{Q}_{max} \in \mathbb{R}^{6n_uN}$;

$$\mathbf{Z}_p = \begin{bmatrix} \mathbf{I}_u & \mathbf{0} & \mathbf{0} \\ -\mathbf{I}_u & \mathbf{0} & \mathbf{0} \\ -\mathbf{I}_u & \mathbf{I}_u & \mathbf{0} \\ \mathbf{I}_u & -\mathbf{I}_u & \mathbf{0} \\ \mathbf{I}_u & -2\mathbf{I}_u & \mathbf{I}_u \\ -\mathbf{I}_u & 2\mathbf{I}_u & \mathbf{I}_u \end{bmatrix}, \quad \text{with: } p = 0, \dots, N-3\,,\ \mathbf{I}_u \in \mathbb{R}^{n_u},\ \mathbf{Z}_p \in \mathbb{R}^{6n_u \times 3n_u},$$

$$\mathbf{z}_{N-2} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{I}_u \\ -\mathbf{I}_u \end{bmatrix}, \quad \mathbf{Z}_{N-2} = \begin{bmatrix} \mathbf{0} & \mathbf{I}_u & \mathbf{0} \\ \mathbf{0} & -\mathbf{I}_u & \mathbf{0} \\ -\mathbf{I}_u & \mathbf{I}_u & \mathbf{0} \\ \mathbf{I}_u & -\mathbf{I}_u & \mathbf{0} \\ -2\mathbf{I}_u & \mathbf{I}_u & \mathbf{0} \\ 2\mathbf{I}_u & -\mathbf{I}_u & \mathbf{0} \end{bmatrix}, \quad \mathbf{Z}_{N-1} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{I}_u \\ \mathbf{0} & \mathbf{0} & -\mathbf{I}_u \\ \mathbf{0} & -\mathbf{I}_u & \mathbf{I}_u \\ \mathbf{0} & \mathbf{I}_u & -\mathbf{I}_u \\ \mathbf{I}_u & -2\mathbf{I}_u & \mathbf{I}_u \\ -\mathbf{I}_u & 2\mathbf{I}_u & -\mathbf{I}_u \end{bmatrix}$$

in which: $\mathbf{z}_{N-2} \in \mathbb{R}^{6n_u}$, $\mathbf{Z}_{N-2} \in \mathbb{R}^{6n_u \times 3n_u}$, $\mathbf{Z}_{N-1} \in \mathbb{R}^{6n_u \times 3n_u}$, and where:

$$\mathbf{q}_{max_p} = \mathbf{q}_{max_{N-1}} = \mathbf{q}_{max_{N-2}} = \begin{bmatrix} \tilde{\mathbf{u}}_{max} \\ \tilde{\mathbf{u}}_{min} \\ (\frac{d\tilde{\mathbf{u}}}{dt})_{max} \cdot t_s \\ (\frac{d\tilde{\mathbf{u}}}{dt})_{min} \cdot t_s \\ (\frac{d^2\tilde{\mathbf{u}}}{dt^2})_{max} \cdot t_s^2 \\ (\frac{d^2\tilde{\mathbf{u}}}{dt^2})_{min} \cdot t_s^2 \end{bmatrix}, \quad \begin{array}{c} \text{with: } p = 0, \dots, N-3, \\ \mathbf{q}_{max_p} \in \mathbb{R}^{6n_u} \end{array}$$

Equality constraints

Equality constraints are imposed in order to guarantee continuity and avoid large and rapid variations in the control variables when the learning trajectory starts or ends. Constraints are imposed for $k = 0$ and $k = N-1$: $\tilde{\mathbf{u}}_j(0) = \bar{\tilde{\mathbf{u}}}_0$ and $\tilde{\mathbf{u}}_j(N-1) = \bar{\tilde{\mathbf{u}}}_{N-1}$. In lifted form it becomes:

$$\mathbf{Z}_{eq}\mathbf{U}_j = \mathbf{Q}_{eq}, \tag{2.82}$$

where:

$$\mathbf{Z}_{eq} = \text{diag}(\mathbf{Z}_{eq_0}, \dots, \mathbf{Z}_{eq_i}, \dots, \mathbf{Z}_{eq_{N-1}}), \qquad \mathbf{Z}_{eq} \in \mathbb{R}^{n_u N \times n_u N},$$

$$\mathbf{Z}_{eq_i} = \begin{cases} \mathbf{I}_u & \text{if } i = (0, N-1) \\ \mathbf{0} & \text{elsewhere} \end{cases}, \qquad \mathbf{Z}_{eq_i} = \in \mathbb{R}^{n_u \times n_u},$$

and with:

$$\mathbf{Q}_{eq} = \begin{bmatrix} \mathbf{q}_{eq_0}^T & \cdots & \mathbf{q}_{eq_i}^T \cdots \mathbf{q}_{eq_{N-1}}^T \end{bmatrix}^T, \qquad \mathbf{Q}_{eq} \in \mathbb{R}^{n_u N \times n_u N},$$

$$\mathbf{q}_{eq_i} = \begin{cases} \bar{\mathbf{u}}_i & \text{if } i = (0, N-1) \\ \mathbf{0} & \text{elsewhere} \end{cases}, \qquad \mathbf{q}_{eq} \in \mathbb{R}^{n_u}.$$

These 'hard' constraints are, as explained, additional design parameters that influence the optimization problem and the input update. Table 2.3 displays the constraints used where, for the symmetry of the problem under study, the following relations have been imposed:

$$\begin{aligned} \bar{\mathbf{u}}_l &= \tilde{\mathbf{u}}_{max} = \tilde{\mathbf{u}}_{min} \\ \bar{\mathbf{u}}_{d1} &= \left(\frac{d\tilde{\mathbf{u}}}{dt}\right)_{max} = \left(\frac{d\tilde{\mathbf{u}}}{dt}\right)_{min} \\ \bar{\mathbf{u}}_{d2} &= \left(\frac{d^2\tilde{\mathbf{u}}}{dt^2}\right)_{max} = \left(\frac{d^2\tilde{\mathbf{u}}}{dt^2}\right)_{min} . \end{aligned}$$

Design Parameters	
• **Kalman**:	$\hat{\mathbf{D}}_0, \mathbf{P}_0, \mathbf{\Sigma}_j, \mathbf{H}_j$
• **Weights**:	$\mathbf{W}_e,\ \mathbf{W}_{\Delta u},\ \mathbf{W}_u,\ \mathbf{W}_v$
• **Constraints**:	
- Inequality:	$\mathbf{Z}\mathbf{U}_{j+1} \leq \mathbf{Q}_{max}$
Limits	$-\bar{\mathbf{u}}_l \leq \tilde{\mathbf{u}}(k) \leq \bar{\mathbf{u}}_l$
1^{st} der. Limits	$-\bar{\mathbf{u}}_{d1} \leq \frac{\Delta\tilde{\mathbf{u}}}{\Delta k} \leq \bar{\mathbf{u}}_{d1}$
2^{nd} der. Limits	$-\bar{\mathbf{u}}_{d2} \leq \frac{\Delta^2\tilde{\mathbf{u}}}{\Delta k^2} \leq \bar{\mathbf{u}}_{d2}$
- Equality:	$\mathbf{Z}_{eq}\mathbf{U}_{j+1} = \mathbf{Q}_{eq}$
Initial	$\tilde{\mathbf{u}}(0) = \bar{\mathbf{u}}_i$
Final	$\tilde{\mathbf{u}}(N-1) = \bar{\mathbf{u}}_f$

Table 2.3: K-ILC design parameters - constraints

2.6 K-ILC design parameters

Table 2.4 shows all the matrices and vectors parameters that need to be chosen to improve the performance of the K-ILC.

As discussed in the estimation step (for the Kalman parameters) and in the learning update step (for the design parameters that regulates the optimization problem), the many free parameters can be lowered to just five scalar coefficients plus five vector constraints. Among the five scalar terms, two couples, namely $\bar{\sigma}$ and $\bar{\eta}$, and w_e and $w_{\Delta u}$ are closely related to one another. These free parameter chosen are reported in Table 2.5.

Design Parameters	
• **Kalman**:	$\hat{\mathbf{D}}_0,\ \mathbf{P}_0, \mathbf{\Sigma}_j, \mathbf{H}_j$
• **Weights**:	$\mathbf{W}_e,\ \mathbf{W}_{\Delta u},\ \mathbf{W}_u,\ \mathbf{W}_v$
• **Constraints**:	
- Inequality:	$\mathbf{Z},\ \mathbf{Q}_{max}$
- Equality:	$\mathbf{Z}_{eq},\ \mathbf{Q}_{eq}$

Table 2.4: K-ILC design parameters

Design Parameters	
• **Kalman**:	$p_0, \bar{\sigma}, \bar{\eta}$
• **Weights**:	$w_e,\ w_{\Delta u}$
• **Constraints**:	
- Inequality:	$\bar{\mathbf{u}}_l,\ \bar{\mathbf{u}}_{d1}, \bar{\mathbf{u}}_{d2}$
- Equality:	$\bar{\mathbf{u}}_i,\ \bar{\mathbf{u}}_f$

Table 2.5: K-ILC scalar design parameters

2.7 K-ILC vs Q-ILC

The Kalman-ILC can be considered as an extension of a Quadratic-ILC [6]. In other terms, the K-ILC results in a Q-ILC when specific conditions are met.

Q-ILC uses the most straightforward method to predict the error. In fact, the estimated error of the next iteration, $\hat{\mathbf{E}}_{j+1}$, is supposed to be equal to the current error measurement $\mathbf{e}_j$, plus the contribution of the nominal model predicted error change:

$$\hat{\mathbf{E}}_{j+1}^{\text{Q-ILC}}(\mathbf{U}_{j+1}) = \mathbf{E}_j + \mathbf{F}\Delta\mathbf{U}_{j+1}, \tag{2.83}$$

where the last term is obtained from the nominal error definition in equation (2.27): $\tilde{\mathbf{E}}_{j+1} - \tilde{\mathbf{E}}_j = \mathbf{F}(\mathbf{V}_{j+1} - \mathbf{V}_j) = \mathbf{F}\Delta\mathbf{U}_{j+1}$[8].

Recalling the definition in equation (2.53), the error prediction of the K-ILC can, instead, be rewritten as follows:

$$\begin{aligned}\hat{\mathbf{E}}_{j+1}^{\text{K-ILC}}(\mathbf{U}_{j+1}) &= \mathbf{F}\mathbf{V}_{j+1} + \hat{\mathbf{D}}_{j+1} && (2.84)\\ &= \mathbf{F}(\mathbf{V}_{j+1} \pm \mathbf{V}_j) + \hat{\mathbf{D}}_{j+1} && (2.85)\\ &= \mathbf{F}\Delta\mathbf{V}_{j+1} + \mathbf{F}\mathbf{V}_j + \hat{\mathbf{D}}_{j+1}. && (2.86)\end{aligned}$$

[8]This expression is analogous to the one reported in the previous chapter except for the change in sign in the error definition. Specifically, $\tilde{\mathbf{P}}$ is replaced with $\mathbf{F}$.

Since: $\Delta\mathbf{U}_{j+1} = \Delta\mathbf{V}_{j+1}$ (with $\Delta\mathbf{U}_{j+1} := \mathbf{U}_{j+1} - \mathbf{U}_j$ and $\Delta\mathbf{V}_{j+1} := \mathbf{V}_{j+1} - \mathbf{V}_j$), equations (2.83) and (2.86) are the same when:

$$\mathbf{E}_j = \mathbf{F}\mathbf{V}_j + \hat{\mathbf{D}}_{j+1}.$$

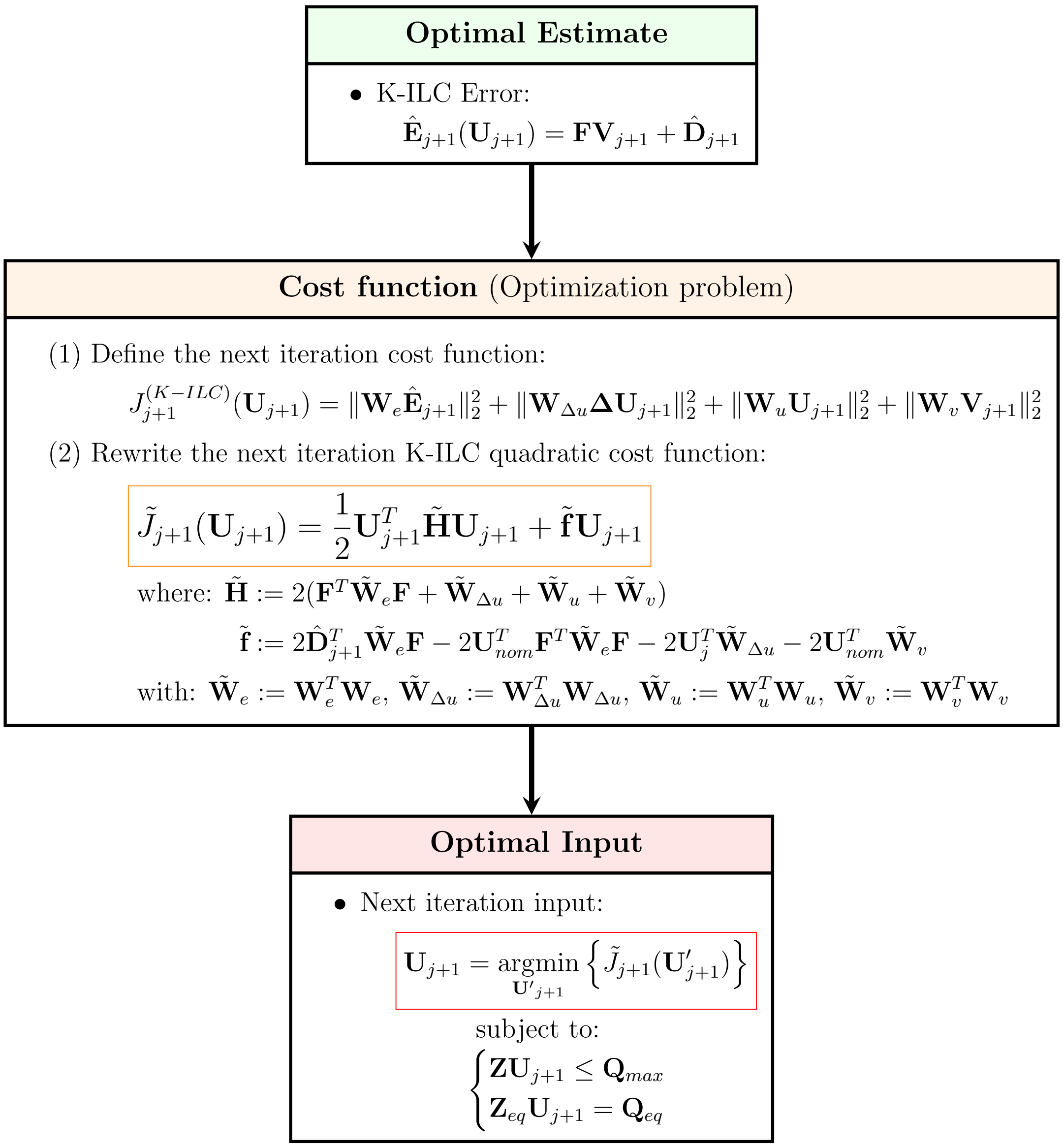
This is the case when $\hat{\mathbf{D}}_{j+1} = \mathbf{E}_j - \mathbf{F}\mathbf{V}_j$ as in equation (2.44). In fact, if the noise in measurements is considered negligible, the Kalman gain approaches the identity matrix. When $\mathbf{K}_j = \mathbf{I}$, the next iteration disturbance prediction is exclusively determined by the current-iteration error measurement and the current input shift as pointed out in equation (2.44).

To summarize, if the measurement are completely trusted, *i.e.*, if the Kalman gain is equal to the identity matrix, the following holds:

$$\hat{\mathbf{E}}^{\text{K-ILC}}_{j+1}(\mathbf{U}_{j+1}) = \hat{\mathbf{E}}^{\text{Q-ILC}}_{j+1}(\mathbf{U}_{j+1}) = \mathbf{E}_j + \mathbf{F}\Delta\mathbf{U}_{j+1}, \tag{2.87}$$

therefore, if the optimization problem for the input update is the same, the K-ILC and the Q-ILC are identical.

Lastly, a graphical representation is shown for a better understanding: Figure 2.6 and 2.7 illustrate respectively the block diagrams of the K-ILC and the Q-ILC . Once again, when the measurement are solely trusted and when the optimization setup is equivalent ($\tilde{\mathbf{H}} = \tilde{\mathbf{H}}_Q$, $\tilde{\mathbf{f}} = \tilde{\mathbf{f}}_Q$), the two block diagrams describe the same process.



Optimal Estimate

- K-ILC Error:

$$\hat{\mathbf{E}}_{j+1}(\mathbf{U}_{j+1}) = \mathbf{F}\mathbf{V}_{j+1} + \hat{\mathbf{D}}_{j+1}$$

Cost function (Optimization problem)

(1) Define the next iteration cost function:

$$J^{(K-ILC)}_{j+1}(\mathbf{U}_{j+1}) = \|\mathbf{W}_e\hat{\mathbf{E}}_{j+1}\|_2^2 + \|\mathbf{W}_{\Delta u}\boldsymbol{\Delta}\mathbf{U}_{j+1}\|_2^2 + \|\mathbf{W}_u\mathbf{U}_{j+1}\|_2^2 + \|\mathbf{W}_v\mathbf{V}_{j+1}\|_2^2$$

(2) Rewrite the next iteration K-ILC quadratic cost function:

$$\boxed{\tilde{J}_{j+1}(\mathbf{U}_{j+1}) = \frac{1}{2}\mathbf{U}^T_{j+1}\tilde{\mathbf{H}}\mathbf{U}_{j+1} + \tilde{\mathbf{f}}\mathbf{U}_{j+1}}$$

where: $\tilde{\mathbf{H}} := 2(\mathbf{F}^T\tilde{\mathbf{W}}_e\mathbf{F} + \tilde{\mathbf{W}}_{\Delta u} + \tilde{\mathbf{W}}_u + \tilde{\mathbf{W}}_v)$

$\tilde{\mathbf{f}} := 2\hat{\mathbf{D}}^T_{j+1}\tilde{\mathbf{W}}_e\mathbf{F} - 2\mathbf{U}^T_{nom}\mathbf{F}^T\tilde{\mathbf{W}}_e\mathbf{F} - 2\mathbf{U}^T_j\tilde{\mathbf{W}}_{\Delta u} - 2\mathbf{U}^T_{nom}\tilde{\mathbf{W}}_v$

with: $\tilde{\mathbf{W}}_e := \mathbf{W}^T_e\mathbf{W}_e$, $\tilde{\mathbf{W}}_{\Delta u} := \mathbf{W}^T_{\Delta u}\mathbf{W}_{\Delta u}$, $\tilde{\mathbf{W}}_u := \mathbf{W}^T_u\mathbf{W}_u$, $\tilde{\mathbf{W}}_v := \mathbf{W}^T_v\mathbf{W}_v$

Optimal Input

- Next iteration input:

$$\boxed{\mathbf{U}_{j+1} = \underset{\mathbf{U'}_{j+1}}{\operatorname{argmin}}\left\{\tilde{J}_{j+1}(\mathbf{U}'_{j+1})\right\}}$$

subject to:

$$\begin{cases}\mathbf{Z}\mathbf{U}_{j+1} \leq \mathbf{Q}_{max}\\ \mathbf{Z}_{eq}\mathbf{U}_{j+1} = \mathbf{Q}_{eq}\end{cases}$$

Figure 2.5: Learning step of the K-ILC

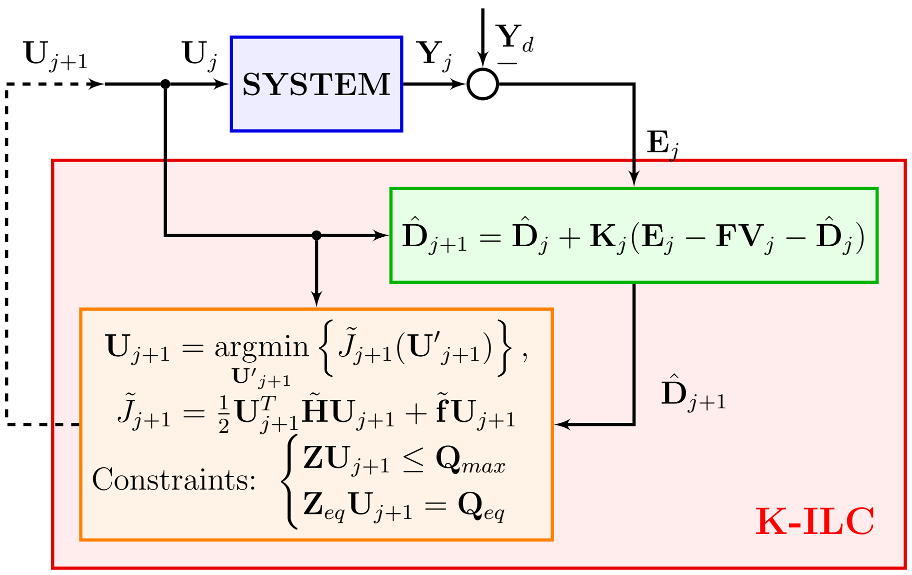

$\mathbf{U}_{j+1}$
$\mathbf{U}_j$
SYSTEM
$\mathbf{Y}_j$
$\mathbf{Y}_d$
$-$
$\mathbf{E}_j$
$\hat{\mathbf{D}}_{j+1} = \hat{\mathbf{D}}_j + \mathbf{K}_j(\mathbf{E}_j - \mathbf{F}\mathbf{V}_j - \hat{\mathbf{D}}_j)$
$\mathbf{U}_{j+1} = \underset{\mathbf{U}'_{j+1}}{\text{argmin}} \left\{ \tilde{J}_{j+1}(\mathbf{U}'_{j+1}) \right\},$
$\tilde{J}_{j+1} = \frac{1}{2}\mathbf{U}^T_{j+1}\tilde{\mathbf{H}}\mathbf{U}_{j+1} + \tilde{\mathbf{f}}\mathbf{U}_{j+1}$
Constraints: $\begin{cases} \mathbf{Z}\mathbf{U}_{j+1} \leq \mathbf{Q}_{max} \\ \mathbf{Z}_{eq}\mathbf{U}_{j+1} = \mathbf{Q}_{eq} \end{cases}$
$\hat{\mathbf{D}}_{j+1}$
K-ILC

Figure 2.6: K-ILC block diagram in lifted form

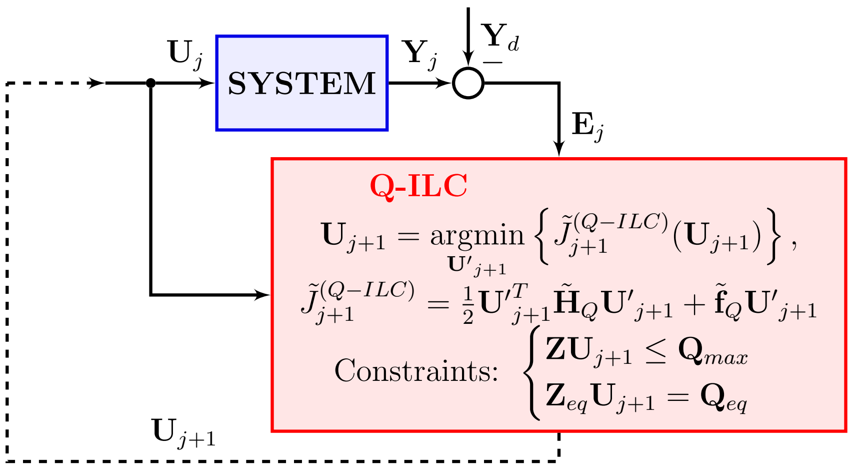

$\mathbf{U}_j$
SYSTEM
$\mathbf{Y}_j$
$\mathbf{Y}_d$
$-$
$\mathbf{E}_j$
Q-ILC
$\mathbf{U}_{j+1} = \underset{\mathbf{U}'_{j+1}}{\text{argmin}} \left\{ \tilde{J}^{(Q-ILC)}_{j+1}(\mathbf{U}_{j+1}) \right\},$
$\tilde{J}^{(Q-ILC)}_{j+1} = \frac{1}{2}\mathbf{U}'^T_{j+1}\tilde{\mathbf{H}}_Q\mathbf{U}'_{j+1} + \tilde{\mathbf{f}}_Q\mathbf{U}'_{j+1}$
Constraints: $\begin{cases} \mathbf{Z}\mathbf{U}_{j+1} \leq \mathbf{Q}_{max} \\ \mathbf{Z}_{eq}\mathbf{U}_{j+1} = \mathbf{Q}_{eq} \end{cases}$
$\mathbf{U}_{j+1}$

Figure 2.7: Q-ILC block diagram in lifted form

Chapter 3

Implementation of a K-ILC to a multirotor UAV

This chapter contains the description of the Simulink model and the MATLAB code developed to test the performance of a Kalman iterative learning control for a quadrotor Unmanned Air Vehicle (UAV). An overview of the design process in terms of the general architecture and the implementation steps followed is illustrated. The K-ILC is first implemented in a general feedback SISO system and then applied to a MIMO system. Once the effectiveness of the control action is validated, the iterative control is installed in a pre-existent quadcopter model. The Simulink K-ILC quadcopter model and the MATLAB code structure of the simulation are detailed. Lastly the quadrotor simulation results are presented and the choice of the main design parameters used is discussed.

3.1 Implementation steps

This section describes the engineering design process, meaning the series of steps followed, to come up with a solution to the problem of improving the performance of an autonomous quadrotor vehicle in precisely tracking a pre-set trajectory using a K-ILC. The goal is to teach a quadrotor how to achieve an high tracking performance in executing a specific manoeuvre when model errors and repetitive disturbances are present. This is done by designing a K-ILC algorithm which is able to adapt the feed-forward input signal entering the system on the basis of past trial performance of the system.

The approach presented is based on the norm-optimal Kalman iterative learning control algorithm described in the previous chapter.

3.1.1 Implementation environment and architecture

The first design step to implement a new control law in a pre-existent system (in a quadrotor) is the formalization of the environment and the architectural choices that concur to shape the problem.

In this thesis, MATLAB® and Simulink® are used to explore the Kalman iterative learning control design, given that both software can generate code (in C, C++ and other languages) which can be exported and integrated into external applications and devices.

MATLAB is a proprietary multi-paradigm programming language and numerical computing environment developed by MathWorks that allows for numerous applications including matrix manipulations, implementation of algorithms, data plotting, creation of user interfaces, etc.

Simulink is a MATLAB-based graphical programming environment used for modeling, simulating and analyzing multi-domain dynamical systems. Thanks to a set of customizable block libraries and the graphical block diagramming tool, that allows to model a dynamical system as a block diagram simply using drag-and-drop features provided by the GUI, it enables rapid construction and simulation of virtual prototypes to explore design concepts.

Since it offers tight integration with the rest of the MATLAB environment, the 'actual' dynamical system model (`SYSTEM` block) is implemented and simulated in Simulink (Figure 3.1).

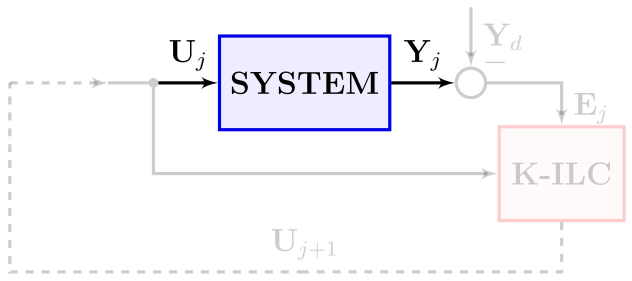


Figure 3.1: Simulink® implementation

The Kalman iterative learning control is, instead, coded in MATLAB (Figure 3.2) given that the controller works 'off-line', which means that it is activated after each simulation is completed, *i.e.*, once all the data gathered during the simulation is available.

As previously mentioned, in both block diagrams (3.1 and 3.2), the signals depicted are in lifted form. In the practical implementation, the `SYSTEM` block works with (digital) discrete-time input signals using *time series* objects, *i.e.* data vectors sampled over time.

Note that the 'physical' system built in Simulink in which the K-ILC is implemented differs from the approximated model used to define the K-ILC algorithm.

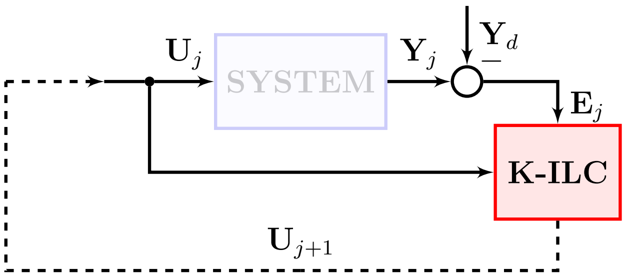


Figure 3.2: MATLAB® implementation

In fact, the K-ILC algorithm requires a linear dynamical model of the (physical) system under study. This approximated nominal model serves:

- to obtain an initial guess of the feed-forward input given a desired trajectory;
- to provide the direction for feed-forward corrections in the input update step.

The discrete linear nominal system in state space representation (*i.e.* matrices $\mathbf{A}_d$, $\mathbf{B}_d$, $\mathbf{C}_d$), is estimated directly in Simulink, exploiting the System Identification Toolbox. This app allows to construct the (linear) mathematical model that approximate the dynamic system from measured input-output data. In practise, the `SYSTEM` model is treated as an unknown model using a '*black-box*' identification strategy in order to obtain the linear nominal lifted model matrix $\mathbf{F}$ linking the input vector $\mathbf{U}_j$ to the output $\mathbf{Y}_j$ (aside from model errors, external disturbances and noises).

3.1.2 K-ILC applied to a SISO system

Instead of working directly with the multi-variable quadcopter model, the K-ILC is first implemented in a known single input, single output system.

The 'actual' model is built in Simulink (Figure 3.3).

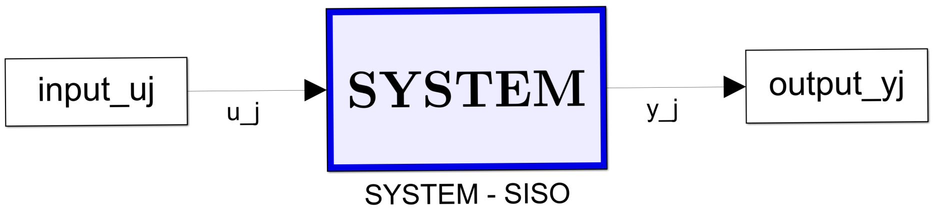


Figure 3.3: Simulink® - SISO system

The blocks in Figure 3.3 are listed below:

`input_uj`: the input block is a *time-series* object defined in the MATLAB workspace that represent the continuous-time evolution of the discrete input signal generated by the Kalman iterative learning control. Through interpolation, it generates a ‘continuous’ signal `u_j`[1] that enter the SISO system.

`input_yj`: it is also a *time series* object obtained sampling (with the same sample frequency with which the K-ILC works) the ‘continuous’ signal `y_j` exiting the system.

`SYSTEM`: the system block, defined in the continuous time-domain, represents a feedback asymptotically stable dynamical system (Figure 3.4). The plant of

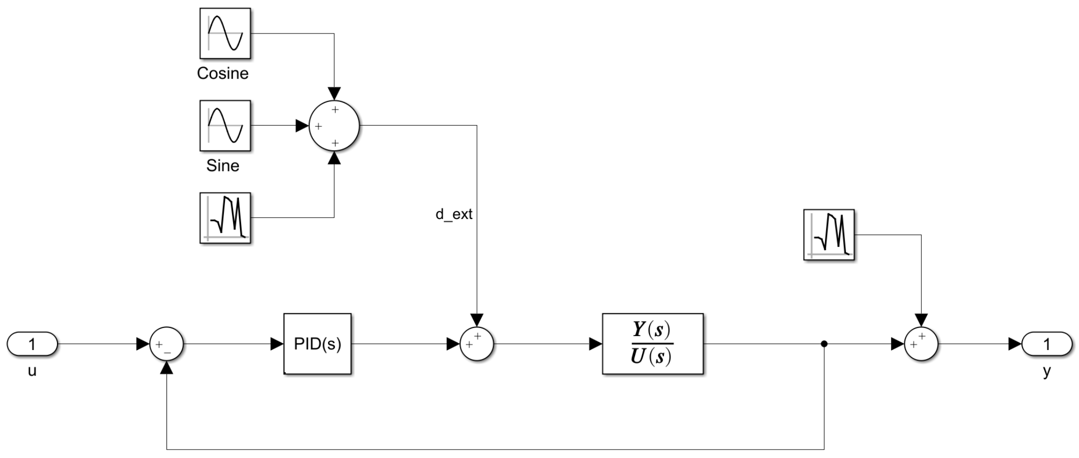


Figure 3.4: Simulink® - feedback SISO system

the SISO system is a second order transfer function stabilized by a PID feedback controller. External disturbances, which enter the plant and perturb the process, include both repetitive and non-repetitive terms (with respect to iterations). The repetitive disturbances are modeled in terms of sinusoidal and co-sinusoidal signals with different frequencies, whereas non-repetitive contributions are modeled as noise (random term).

The K-ILC algorithm is coded in the MATLAB environment in accordance to the methodology described in the previous chapter[2].

The aforementioned nominal linear model, in which the K-ILC is applied, is obtained from the approximation of the ‘actual’ SISO system built in Simulink.

[1]Actually Simulink works only with digital, *i.e.* discrete, signals. The step size of each simulation is however chosen in order to keep the time quantization error small enough on the basis of the defined signal-domain.

[2]The structure of the code is similar to the one presented in the following for the UAV implementation.

Specifically, the transfer function defining the input-output relation is determined directly from the System Identification Toolbox in Simulink, introducing a step input and corresponding step output (that is the measured system step response) time-series. The closed-loop transfer function is then converted to the 'continuous' state-space representation, which is in turn discretized and transformed in lifted form.

Through this simplified implementation (the 'real' model is SISO and completely defied also in term of the external disturbances), it is possible to play with the different K-ILC design parameters and to analyse the sensitivity of the system performance to the variations of these parameters, so as to select them properly. In addition, the performance of the ILC system is validated also in presence of model (multiplicative) uncertainties to ensure the robustness of the K-ILC design.

A qualitative solution (actually based on one of the many simulation) is presented in Figure 3.5.

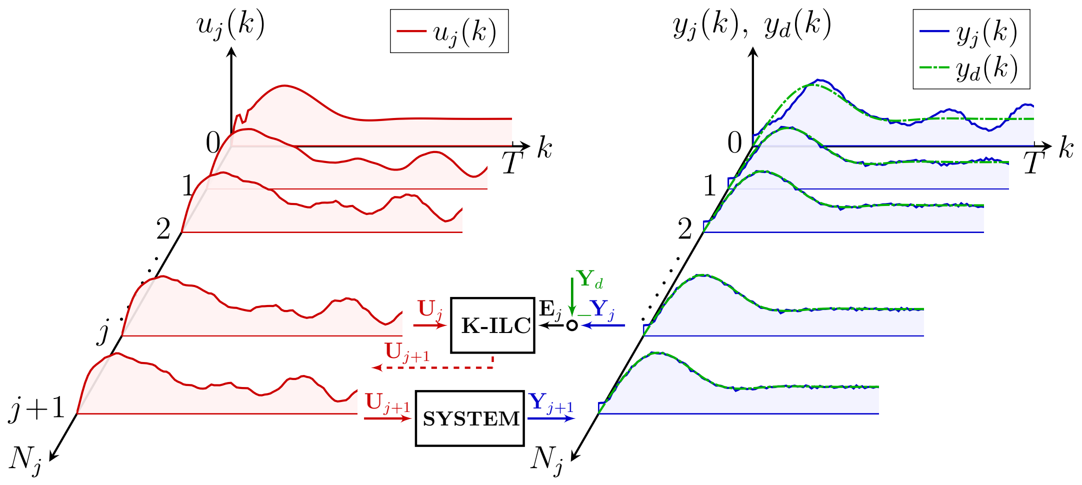


Figure 3.5: K-ILC SISO system qualitative results

As it is evident in Figure 3.5, the K-ILC input signal (in red) enters the system and gives an output (blue line) that differs from the desired output (green-dashed line). The K-ILC is able to understand, based on past iterations, the new signal that compensate for repetitive disturbances and identification model errors. In fact, the effects of repetitive known sinusoidal terms (repetitive external disturbances) are gradually removed from the output solution trough a proper input signal update. The same is true for the known[3] nominal model error.

[3]Since this 'actual' model is known and the nominal one is identified (considering the SYSTEM block as black-box), the error between the two models is also known.

3.1.3 K-ILC applied to a MIMO system

Once the effectiveness of the K-ILC has been validated for the SISO system, the iterative learning control strategy was extended to the multi-input, multi-output system.

Figure 3.6 shows the 'real' model built in Simulink. The MIMO system considered is a stable feedback system with two inputs and two outputs.

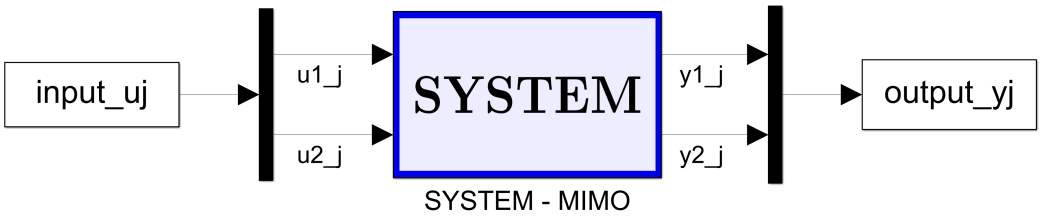


Figure 3.6: Simulink® - MIMO system

`SYSTEM` block is displayed in Figure 3.7. The plant of the system is described using the state-space representation. The close-loop linear dynamic system is asymptotically stable, *i.e.* all the poles of the (coupled matrix) transfer function are in the open left-half complex plane. Analogously to the SISO system, the disturbances, both repetitive (sinusoidal and co-sinusoidal) and non-repetitive (noise) enter directly the plant.

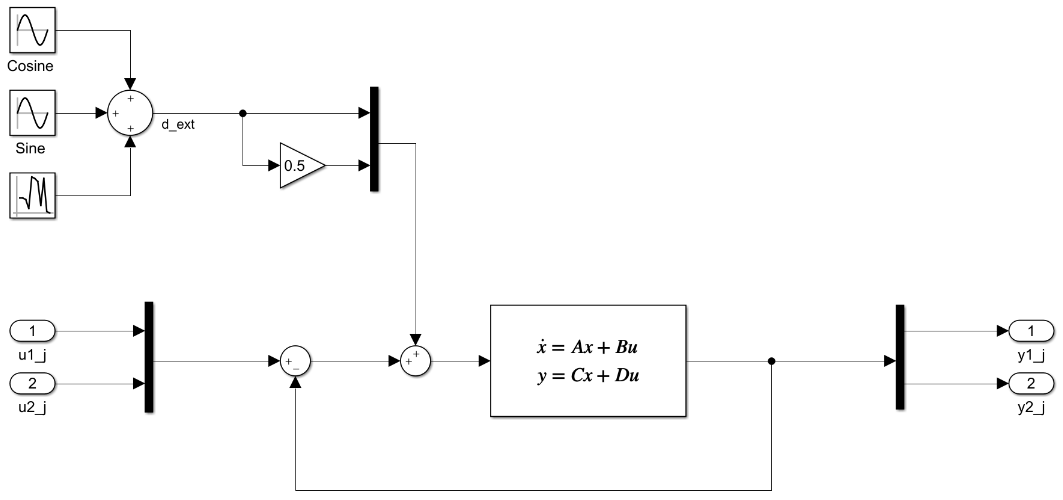


Figure 3.7: Simulink® - feedback MIMO system

Also in this case, the nominal lifted model ($\tilde{\mathbf{Y}}_j = \mathbf{F}\mathbf{U}_j$) is identified from the input-output data using the Simulink identification toolbox. As for the SISO system, the MIMO model is treated as a '*black-box*', meaning that the `SYSTEM` block is supposed to be unknown. Applying a single step input one at a time, given the output measurement time-series, it is possible to construct the feedback system transfer function matrix, form which a state-space description is obtained; the linear system is thus discretized and converted in lifted form.

The K-ILC algorithm detailed in the previous chapter is once again coded in MATLAB. This implementation is simply an extension of the SISO K-ILC code for a MIMO system. The K-ILC system gives good results in terms of robustness to the parameter choices, to model uncertainties and to disturbances.

3.1.4 K-ILC applied to a quadrotor UAV

The Kalman iterative learning control is now applied to a quadrotor UAV model. The objective is to improve the quadcopter performance in executing a complex manoeuvrer by precisely tracking a tridimensional trajectory. This is done by acting on the reference trajectory input signal.

The pre-existing model in which the control is applied, is presented in the next subsection. The 'actual' model used for the simulation and its approximation are later described.

3.1.5 Pre-existing model

A Simulink non-linear quadrotor model has been used to validate the implementation of the Kalman iterative learning control.

The pre-existent model[4] of the quadcopter([17] [18]) is displayed in Figure 3.8.

The main blocks that compose the model in Figure 3.8 are the following:

`Controllers`: this block contains the cascaded feedback linearized control loops for position, velocity, acceleration, attitude, and angular rates, as well as the mixer (to convert the thrust and moments control signals in the throttle signals of the four motors). It receives as input the setpoint signal vector, containing the reference positions, velocities, accelerations, jerks, (snaps), attitudes (in quaternions), angular rates and angular accelerations that the quadrotor is supposed to follow and, based on the measurements (of position, velocity, attitude and angular rates), it generates as outputs the percentage throttle of each motor to control the quadrotor.

`Quadrotor`: contains the equations which defines the actual dynamic of the quadrotor; it simulate the quadrotor behaviour by receiving as input the throttle (in percentages) requested by to the four motors.

[4]Actually, the given UAV model has been slightly changed: the `Setpoint` block, which generates the trajectory that the quadrotor is supposed follow, was removed.

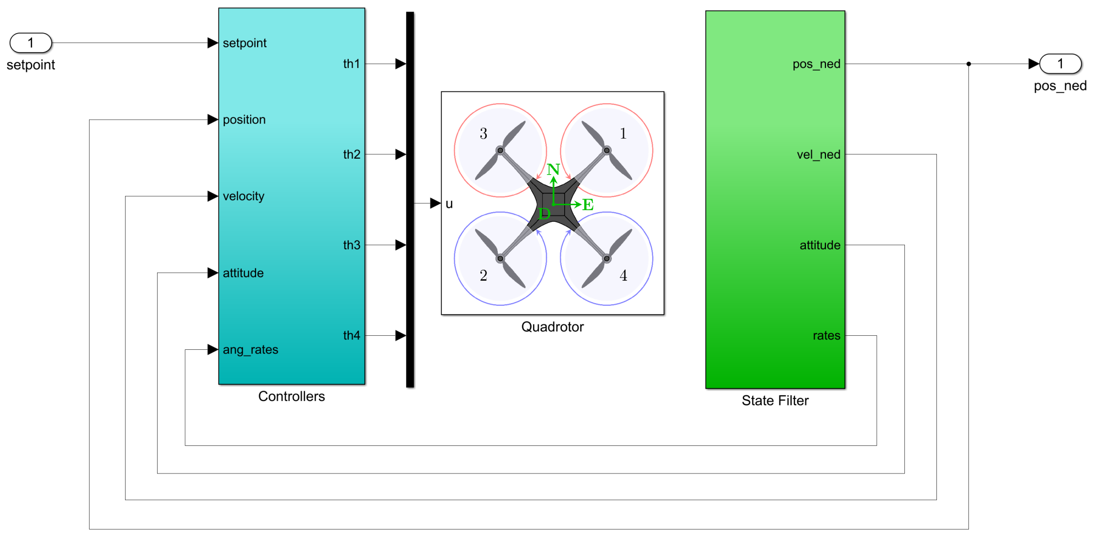


Figure 3.8: Quadcopter model

`State filter`**:** this block reads the outputs of the quadrotor block and transforms the signals by discretizing them and adding noise and delay to mimic measurements.

3.1.6 K-ILC quadcopter model

The implementation of the K-ILC in a quadrotor UAV is analogous to the one developed for the general SISO and MIMO systems. Figure 3.9 shows the Simulink diagram. The blocks in Figure 3.9 are:

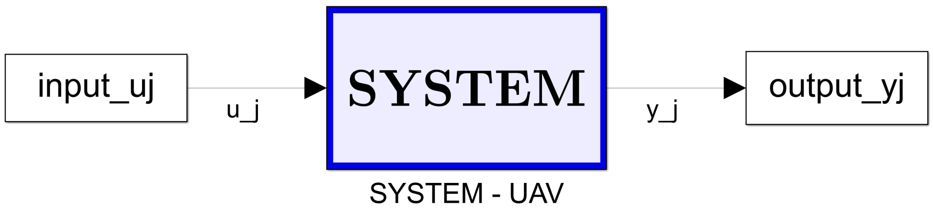


Figure 3.9: Simulink® - UAV system

`input_uj`**:** the input block is a *time-series* object that defines the evolution in time of the quadrotor position, expressed with respect to the local North-East-Down (NED) reference frame. The signal `u_j` (*i.e.* $\mathbf{u}_j(k)$) represents the reference trajectory generated by the Kalman iterative learning control input update law; this discrete signal is re-sampled with the same sampling frequency used by the feedback controller (which is much higher from the one used by the iterative controller).

`input_yj`: it is also a *time-series* object obtained sampling (with the same sample frequency of the iterative controller) the 'continuous' signal `y_j` (*i.e.* $\mathbf{y}_j(t)$) representing the 'actual' position of the drone at the time t.

`SYSTEM`: the system block is displayed in Figure 3.10. It contains the `Setpoint`

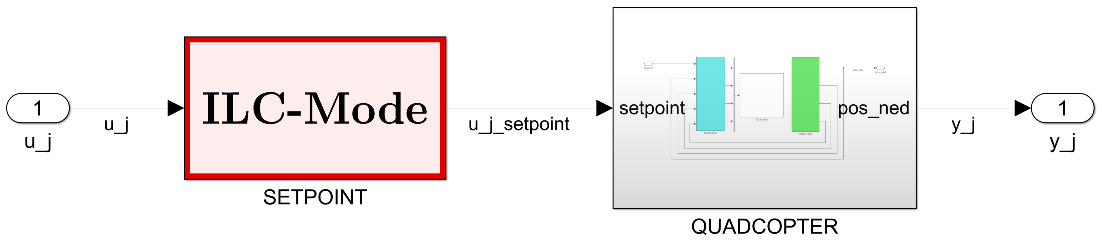


Figure 3.10: Simulink® - UAV system (inner block)

block and the already mentioned quadcopter model. The former block generates the setpoint vector (with 25 signals) that contains the reference values of the feedback controller. Figure 3.11 shows the `Setpoint` block inner structure. Depending on the ILC functioning mode selected by the user, the

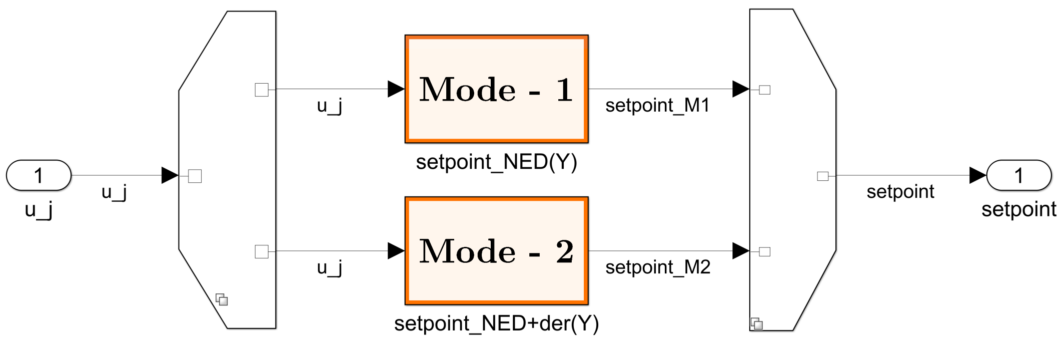


Figure 3.11: ILC-Mode selector

setpoint can varies. When the ILC mode is set to 1, the setpoint entering the quadcopter model does not include feed-forward contributions to the quadcopter controllers, except possibly for the reference yaw signal (when the yaw following toggle is 'on'). The controller can be augmented with feed-forward velocities and accelerations signals, and, optionally yaw feed-forward signals, for known input trajectories, when the ILC mode is set to 2. The architecture of the `Mode-1` block is displayed in Figure 3.12. When the yaw following is enabled, the yaw time-series (of the selected trajectory) defined in the MATLAB workspace (`Yaw_setpoint`) enter the quadcopter model as a feed-forward term (it is before converted in quaternions).

The same is true for the `Mode-2` block in Figure 3.13. In this scheme appears also the blocks `input_uj_der1` and `input_uj_der2` that are the *time-series* objects of the velocity and acceleration feed-foreword signals.

As for the above cases, the K-ILC algorithm is based on the linear time-invariant approximation (nominal model) of the 'actual' dynamical system estimated using the Simulink identification toolbox. The identification procedure used is basic but effective in capturing the key dynamics of the studied system as unmodeled effects are presumed to be compensated by the iterative learning scheme. The nominal model is obtained analysing the output responses to position steps inputs. In practice, a position step input is applied to the drone in the direction of each body axis one at a time. The output measured, *i.e.* the measured position of the drone in the corresponding direction, allows to estimate the input-output transfer functions. The transfer functions are collected and used to contract the decoupled matrix transfer function of the system linking the reference position inputs to the actual output positions (in the NED reference frame). This matrix representing the dynamics of the system is once again used to compute the lifted form nominal model.

3.1.7 Code structure

The K-ILC commented in the prior chapter is coded in MATLAB and simulated in Simulink. The main MATALB script is structured as follows.

- `User choices`: the code specifies the ILC-mode choice (mode 1,2), the yaw following switcher (on/off), the trajectory speed parameter (ω_{traj}), the number of trials (N_j), the initial input choice, the K-ILC design parameters ($\bar{\sigma}$, $\bar{\eta}$, p_0, w_e, $w_{\Delta u}$) and the number of plots displayed.
- `Simulation setup`: the overall K-ILC simulation setup is defined. It includes the following scripts.
  - `Initialization`: the code describes the simulation time samples and the initial conditions (position, velocity, attitude and angular rates).
  - `Trajectory`: trajectory properties are specified in this script; they include the trajectory type (*e.g.* the 8-shape path) the altitude, the scale, the final conditions and the timing of the different flight phases (e.g. take-off time, hovering time, etc.).
  - `ILC setup`: the code defines the ILC timing (activation and deactivation times, sampling time of the ILC), the discrete-time index vectors (*e.g.* k) and the trial lengths (*e.g.* N).
  - `Quadrotor physical parameters`: the script contains the environmental constants, the information of the drone (geometry, mass, structural characteristics, propeller specifications including also aerodynamic

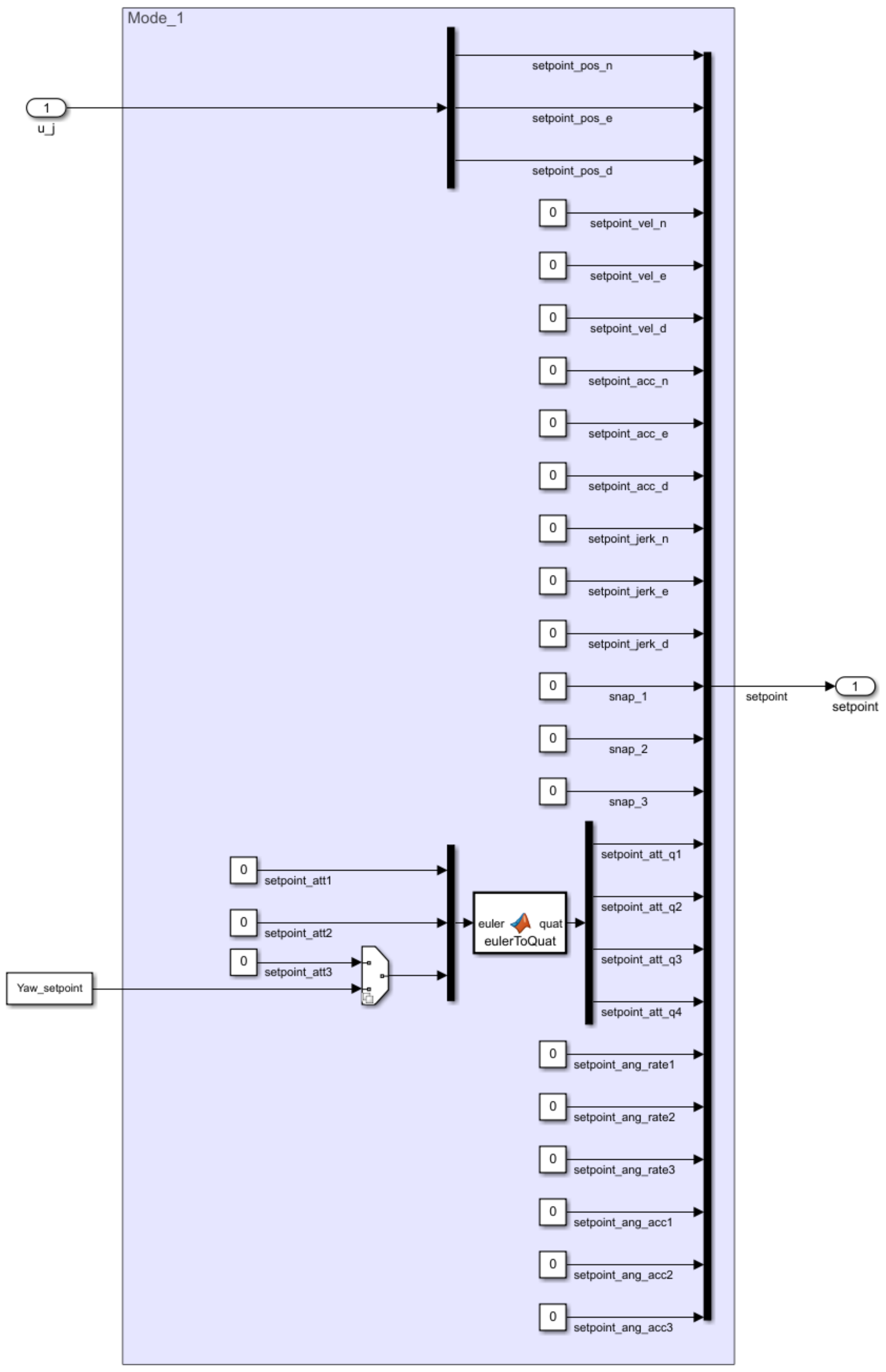


Figure 3.12: Setpoint: Mode-1

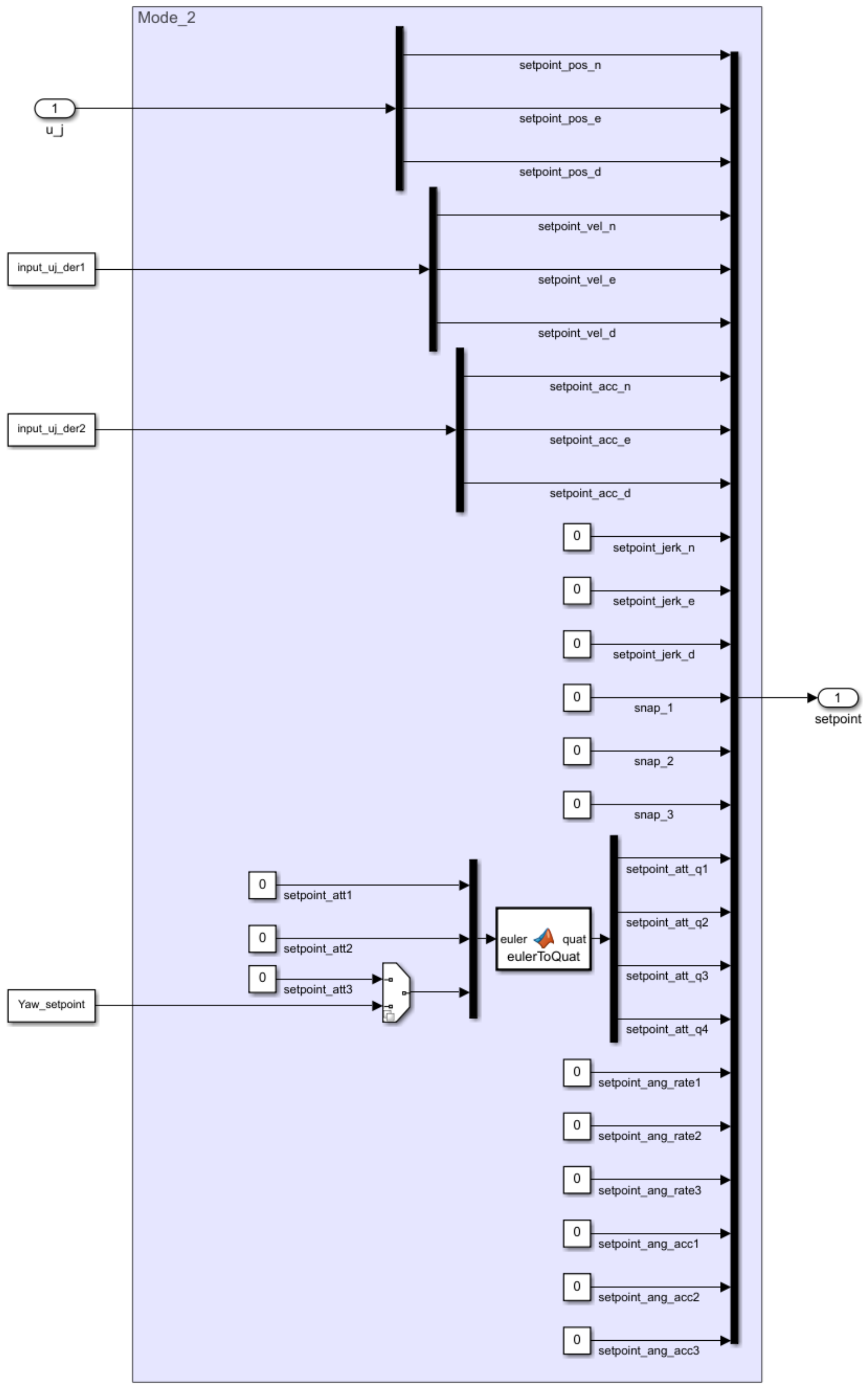


Figure 3.13: Setpoint: Mode-2

data, aerodynamic dumping info and ground effect coefficients), the state estimators properties (noises' standard deviations) and filter delays.

  - `Quadrotor controller parameters`: the script specifies the controller characteristics (sample frequency and time), the saturations (of throttle, thrust and torque) and the regulators parameters (of position, linear-speed, attitude and angular-rate).

- `Model setup`: the models described are distinguished in:
  - *'Actual' model*: all the parameters of the 'real' Simulink model are defined in the MATLAB work-space.
  - *Nominal model*: the lifted representation of the linear discrete state-space system is defined: $\mathbf{F}$ is obtained directly from a function built so as to receives as inputs the state-space matrices $\mathbf{A}_d$, $\mathbf{B}_d$, $\mathbf{C}_d$ and trial length N. These matrices are directly obtained from the identification analysis of the 'actual' Simulink model (using the system identification toolbox).

- `Reference signals`: the script defines the following signals.
  - *Desired output*: through the `setpoint quadrotor generator` script, the desired setpoint in terms of the NED position is computed at each time-instant on the basis of the initial conditions, the timing scheduled and the trajectory properties; the code is completed with the representation of the desired output in lifted form ($\mathbf{Y}_d$) for each sample time considered.
  - *Nominal input*: given the desired output, it is computed the nominal input reported in lifted form ($\mathbf{U}_{nom}$).
  - *Initial input*: the lifted initial input $\mathbf{u}_0$ is determined for the different time sampling considered, choosing between: an input at the initial iteration j equal to the nominal input ($\mathbf{U}_0 = \mathbf{U}_{nom}$), with or without constraint limitations, or a desired output as initial input ($\mathbf{U}_0 = \mathbf{Y}_d$).

- `Constraints`: the script specifies the constraint design parameters for the inequality ($\bar{\mathbf{u}}_l$, $\bar{\mathbf{u}}_{d1}$, $\bar{\mathbf{u}}_{d2}$) and the equality requirements ($\bar{\mathbf{u}}_i$, $\bar{\mathbf{u}}_f$), and their lifted representation.
  - *Inequality constraints*: given the input inequality constraint design parameters ($\bar{\mathbf{u}}_l$, $\bar{\mathbf{u}}_{d1}$, $\bar{\mathbf{u}}_{d2}$), the code computes $\mathbf{Z}$ and $\mathbf{Q}_{\mathbf{max}}$, which are respectively the matrix and the vector of the input inequality constraints in lifted form.

  - *Equality constraints*: analogously, matrix $\mathbf{Z}_{eq}$ and vector $\mathbf{Q}_{eq}$, which represent the equality constraints in lifted form, are computed by the algorithm on the basis of the problem sizes and the design choices ($\bar{\mathbf{u}}_i$, $\bar{\mathbf{u}}_f$).

- `Kalamn parameters and optimization weights`: in this algorithm, the Kalman parameters associated to the initialization of the filter and to the model estimation, and the weights of the input update optimization problem are computed on the basis of the design parameters defined by the user.
  - *Kalman parameters*: the initial Kalman state prediction $\hat{\mathbf{D}}_0$ and the initial variance of the error disturbance prediction $\mathbf{P}_0$, together with the disturbances and error noise covariance matrices ($\mathbf{\Sigma}_j$ and $\mathbf{H}_j$) are obtained form the Kalman design parameter choices (p_0, σ, η).
  - *Optimization weights*: given the selected weights w_e, $w_{\Delta u}$, w_u and w_v, the optimization weight matrices are determined (*i.e.* $\mathbf{W}_e$, $\mathbf{W}_{\Delta u}$, $\mathbf{W}_u$, $\mathbf{W}_v$ and $\tilde{\mathbf{W}}_e$, $\tilde{\mathbf{W}}_{\Delta u}$, $\tilde{\mathbf{W}}_u$, $\tilde{\mathbf{W}}_v$).

- `K-ILC initialization` The script defines the *a priori* setpoint, based on the ILC-mode selected, in terms of position (*i.e.* tracking trajectory), velocity, acceleration, jerk, snap, attitude, angular speed and angular acceleration. The position is defined a priori only for the first iteration, since it changes from iteration to iteration due to the ILC action. The initial simulation ($j = 0$) is run in the 'real' Simulink model so as to obtain the initial reference data of the output, error and input 'shift' required to start the K-ILC for-loop algorithm.

- `K-ILC`: this script represents the main body of the code as it defines the actual Kalman iterative learning control algorithm. The different simulations of the dynamical system in which the ILC is applied are computed recursively, given the input update generated by the controller. Using a for-loop control flow statement (in which the iteration variable j goes from 0 to N_j), the following steps are recursively executed.
  - *Kalman estimation*: the Kalman gain $\mathbf{K}_j$, the state $\hat{\mathbf{D}}_j$ and the variance of the state prediction error $\mathbf{P}_j$ are updated so as to estimate the next-iteration error $\hat{\mathbf{E}}_{j+1}$.
  - *Cost function update*: the cost function to be minimized $\tilde{J}_{j+1}(\mathbf{U}_{j+1})$ is re-evaluated (the adding term $\tilde{\mathbf{f}}_{j+1}$ is computed recursively).
  - *Input update*: using the MATLAB built-in function (`fmincon`) the new input vector is obtained solving the minimization problem with the relative constraints:

```
U_new = fmincon(J_new,U,Z,Q_max,Z_eq,Q_eq),
```

where `J_new` is the next iteration cost function to be minimize, `U_old` is the initial guess chosen to be the the current input vector, `Z`, `Q_max`, `Z_eq`, `Q_eq` defines the inequality and equality constraints, and `U_new` is the next iteration ILC input update.

  – *Run simulation*: the new simulation starts when the `Launch simulator` script is run; this script launches the Simulink model, parses the data collected and plots the drone manoeuvre.
  – *Parsing and saving*: the relevant data of each trial, collected during the whole iteration, is parsed and stored in a 'structure' object.

- `Data sorting`: the information related to the whole K-ILC simulation, including the user choices, the timing schedule, the design parameters selected (Kalman parameters, weights and constraints), the model properties, the simulation specifications (control parameters, drone properties, environment constants, filter noises and delay, saturation limits, trajectory properties and initial conditions selected), the data collected during simulations (including signal vectors $\mathbf{U}_j$, $\mathbf{V}_j$, $\mathbf{Y}_j$, $\mathbf{E}_j$, $\hat{\mathbf{D}}_j$, $\mathbf{U}_{nom}$, $\mathbf{Y}_d$), are cleaned and sorted in a unique structure object.

- `ILC plots`: the script loads the data collected, specifies the plot properties and represents in multiple figures the relevant output plots such as input-output signals for each iteration, the change in tracking trajectories, as well as the cumulative error variations.

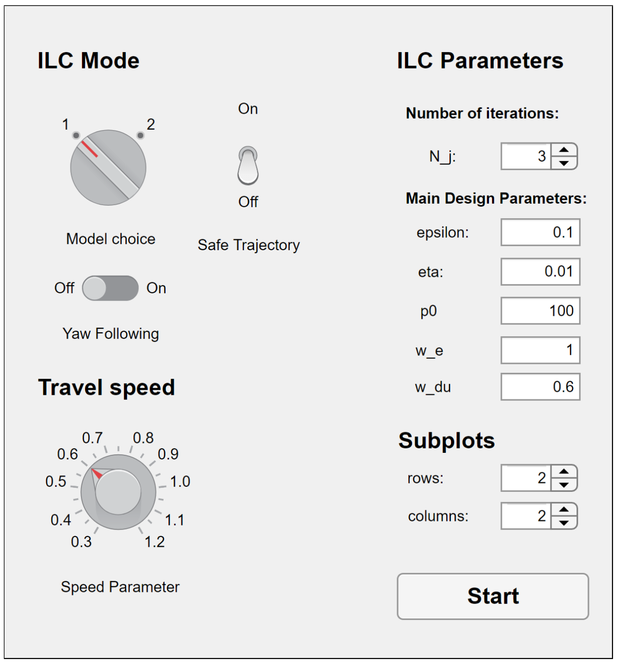
- `Data & figure saving`: the code allows to save the data collected and the figures produced.

3.2 Quadcopter simulation setup

The simulation of the K-ILC quadrotor model depends on numerous parameters. The main simulation parameters can be selected by the user through the graphical user interface (GUI) application built in MATLAB (using the App Designer add-on). The application pane is displayed in the Figure 3.14.

First it is possible to chose the ILC functioning Mode. The ILC-Mode can be set to '1' or '2', varying the model choice knob position. The user can also set the yaw by turning the yaw-following switch 'off' and 'on' respectively to fix the heading or to let it varies during the path[5]. As commented before, the iterative controller implemented in the quadcopter model to track a desired trajectory acts

[5]The additional 'safe trajectory' toggle is always turned 'off' in all the simulations. It is used to analyse if the converged ILC input signal can be used to speed-up the learning of a new trajectory with different time scale (same path, but different travel speed). The problem is not addressed in this thesis



N_j: 3

epsilon: 0.1

eta: 0.01

p0 100

w_e 1

w_du 0.6

rows: 2

columns: 2

Figure 3.14: GUI - Main simulation parameters

exclusively on positions (the ILC provides the input trajectory in North-East-Down coordinates) when the ILC-Mode is set to 1 and the yaw following switch is ‘off’. The quadrotor model can be augmented with velocities, accelerations and attitude feed-forward input signals for known trajectories. This is done by selecting the second ILC-Mode, for velocity and acceleration terms, and turning on the yaw-following switcher to change the yaw during the manoeuvre.

In addition, the travel speed with which the flight path is followed can be varied adjusting the knob to select the ‘speed’ parameter (ω_{traj}).

The user can also modify the main ILC parameters including the number of trials N_j (not including the initial reference trial $j = 0$) and the scalar K-ILC design parameters already commented: $\bar{\sigma}$, $\bar{\eta}$, p_0, w_e and $w_{\Delta u}$.

Lastly the number of rows and columns can be chosen to visualize the simulation.

In the simulation proposed, the quadcopter is controlled by a feedback controller operating at 250 Hz. This frequency differs from the working frequency of the ILC installed, which is chosen to be far lower (10 Hz). The reason for this decision is primarily linked to the time required by the Kalman iterative learning controller to solve the optimization problem for the computation of the input update. In fact, the optimization problem complexity depends on the length of the lifted vectors involved, which in turn is determined by the sampling frequency[6].

[6]It is also a function of the ‘speed’ parameter ω_{traj} since the sampling frequency is fixed: the faster the manoeuvre is performed, the shorter is the vectors’ length.

Since the minimization problem represents a bottleneck of the off-line computational process, the length of the learning time interval is also confined, *i.e.* during take-off, hovering and landing the K-ILC is disabled.

Learning trajectory and flight-path

The manoeuvre to be learned is a periodic eight-shape trajectory flown in the horizontal plane:

$$\begin{cases} x = \frac{1}{2}s \cdot \sin\left(2\omega_{traj}(t - t_i)\right) \\ y = s\cos\left(\omega_{traj}(t - t_i)\right) \\ z = h_s, \end{cases} \tag{3.1}$$

where x, y and z are the coordinates of the NED local reference frame, t is the time variable, t_i is the time in which the eight-shape trajectory starts, s is a scaling coefficient and h_s is the safe altitude. This motion is used to demonstrate the basic working of the K-ILC algorithm.

To avoid big initial state variations, the quadcopter is required to hover at the beginning of the eight-shape trajectory: the learning motion starts and ends in the same hovering point (at the centre of the local frame in North and East position, at a safe altitude h_s). To connect the hovering point to the point in which the eight-shape trajectory starts and to connect as well the point in which the eight-shape trajectory ends to the same hovering point, a 5-th order polynomial interpolation has been considered. The acceleration and de-acceleration phases at the beginning and end of the eight-figure must also be learned precisely. The trajectory shown in Figure 3.15 is then fully defined through the concatenation of a first spline (blue dashed line), the figure-eight path (green line) and second spline (red dashed line).

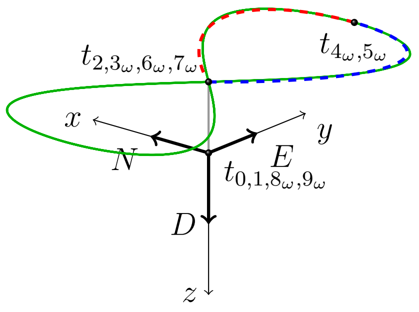

$t_{2,3_\omega,6_\omega,7_\omega}$
$t_{4_\omega,5_\omega}$
x
y
N
E
$t_{0,1,8_\omega,9_\omega}$
D
z

Figure 3.15: Learning trajectory

Each trial simulated consists of the following flight phases.

1. The simulation starts with the multirotor UAV at the origin of the local NED reference frame;

2. at instant t_1 the drone takes-off from the initial position in order to reach a safe altitude;
3. the quadrotor hovers in this position from time-instant t_2 to t_3;
4. at t_3 the K-ILC is activated and the quadrotor moves from the hovering point to the beginning of the 8-shape trajectory (t_{4_ω}[7]), following a five-order interpolation trajectory;
5. the 8-shape trajectory starts at time-instant t_{4_ω} and ends at time-instant t_{5_ω};
6. at t_{6_ω} the drone returns to the initial hovering point (always following a five-order interpolation trajectory) and the ILC is deactivated;
7. the multirotor continues to hover up to time-instant t_{7_ω};
8. from t_{7_ω} to t_{8_ω}, the landing procedure take place and the vehicle regains its initial starting position;
9. at time-instant t_{9_ω} the drone is stopped and the simulation ends.

Learning performance indicator

To analyse the learning performance of the ILC, the following learning performance parameters are defined. A synthetic indicator of the learning performance of the K-ILC algorithm at a specific iteration j is the 'average' position error along the trajectory:

$$e_{pos,j} = \frac{1}{N} \sum_{k=1}^{N} \sqrt{\Delta x_j(k) + \Delta y_j(k) + \Delta z_j(k)} = \frac{1}{N} \|\mathbf{E}_j\|_2, \tag{3.2}$$

where $\Delta x_j(k)$, $\Delta y_j(k)$ and $\Delta z_j(k)$ are defined in $\mathbb{R}$ and denote the deviations of the quadrotor position from the desired trajectory at the discrete time k with respect to the x, y and z axes (of the NED reference frame). Another parameter used to analyse the behaviour of the quadrotor during the different simulations is the adimentional 'average' tracking error, *i.e.*, the 'average' position error along the trajectory at the iteration j divided by the 'average' error computed at the initial iteration ($j = 0$), $\frac{e_{pos,j}}{e_{pos,0}}$. Additionally, to understand how much the 'average' error decreases from two consecutive iterations the following learning performance indicator has been used: $\frac{e_{pos,j} - e_{pos,j-1}}{e_{pos,j}}$.

In order to compare the learning performance of the iterative learning control algorithm, the simulation results presented are referred to the same trajectory (same eight-shape path and same velocity parameter ω_{traj}).

[7]The subscript 'ω' refers to the fact that the time-instant considered is dependent on the ω_{traj} parameter defining the 'speed' with which the trajectory is followed.

3.3 Quadrotor simulation results

The K-ILC algorithm is simulated in MATLAB and Simulink before the real flight experiments are conducted in the real drone. The multiple simulations run have the objective of validating the Kalman iterative learning controller. They aim also to select the best design parameters for the iterative learning control algorithm.

The influence of different design parameters is discussed in section 3.4 The values of the K-ILC parameters used in the simulation presented are reported in Table 3.1.

Symbol	Value	Description
σ	0.1	Process noise scalar parameter
η	0.01	Measurement noise scalar parameter
p_0	100	Initial variance $\mathbf{P}_0$ scalar parameter
w_e	1	Next iteration tracking error scalar weight
$w_{\Delta u}$	0.6	Input change scalar weight
$\bar{u}_l$	4	Max/min input (absolute) constraint coefficient
$\bar{u}_{d1}$	25	Max/min input constraint coefficient of the 1^{st} derivative
$\bar{u}_{d2}$	20	Max/min input constraint coefficient of the 2^{nd} derivative

Table 3.1: Main K-ILC design parameters choice

The first results obtained considering the modeled-based feed-forward velocity and acceleration signals, introduced selecting the ILC-Mode n.2, shows that they do not improve significantly the repeatability of the flight performance. In fact, the ILC compensates for tracking errors almost entirely, as the input signal update also captures the effect of conventional feed-forward terms. For these reasons, in the simulation mentioned the ILC-Mode s set to 1. Additionally, the yaw following is always enabled. The results presented are referred to the eight-shape trajectory previously described considering a 'speed' parameter equal to $\omega_{traj} = 1\,\text{rad/s}$ and a maximum speed below 4.5 m/s. Table 3.2 shows the main simulation parameters used for the simulation considered.

Symbol	Value	Description
ILC-Mode	1	Functioning mode of the K-ILC
Yaw following	'ON'	Yaw following enabler
N_j	25	Number of iterations
ω_{traj}	1 rad/s	Speed parameter
f_c	250 Hz	Feedback controller operating frequency
f_{ILC}	20 Hz	ILC controller operating frequency

Table 3.2: Main simulation parameters choice

In the first iteration it is applied a reference input to the quadcopter model equal to the the desired trajectory. As illustrated in Figure 3.16, the 'actual' quadrotor trajectory (blue line) is far off the desired one (black dashed line) which overlaps the ILC initial input trajectory (red line). The drone trajectory is improved

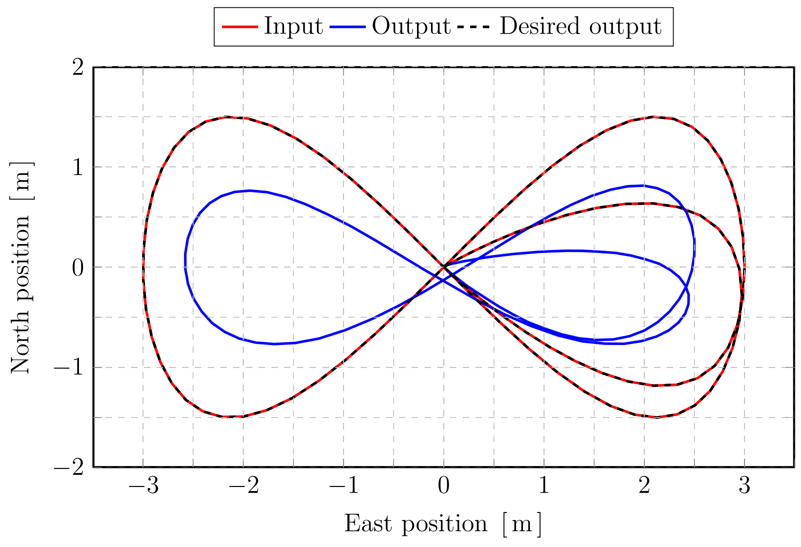


Figure 3.16: Initial iteration ($j = 0$) - 8-shape trajectories

gradually from one iteration to another till reaching convergence.

The NED position setpoint computed by the ILC at convergence and the quadrotor 'real' trajectory is illustrated in Figure 3.17.

The red input trajectory learned by the K-ILC is introduced in the quadrotor as an input and drives it to acceptably track the desired trajectory. Comparing the initial and converged ILC input trajectories (Figures 3.16 and 3.17) is evident how different the shape is. Nevertheless the learned ILC input it is not jittery as it is kept smooth by a proper choice of the optimization parameters (input cost weight and constraints).

Figures 3.18, 3.19, 3.20 illustrate the input, output and reference position signals over time in the North, East and Down axes of the first four iterations (including the initial at $j = 0$) and the last iteration ($j = 25$). As discussed above, during the take-off, the landing and a part of the hovering manoeuvre as well, the ILC is off (yellow background). When the iterative controller is enabled (white background), the data collected in the current iteration during the time interval in which the ILC is on, are used for the next trial input update. It take few trials for the quadrotor to learn the desired manoeuvre.

A first way to analyse the performance of the control algorithm is to compare the initial and final trajectories with the desired one. The quadrotor tracking performance improvement is clearly visible in Figure 3.21. The graph displays the path the drone follows at the initial iteration (dark blue line) and at convergence (light blue line) compared to the reference path (black dashed line).

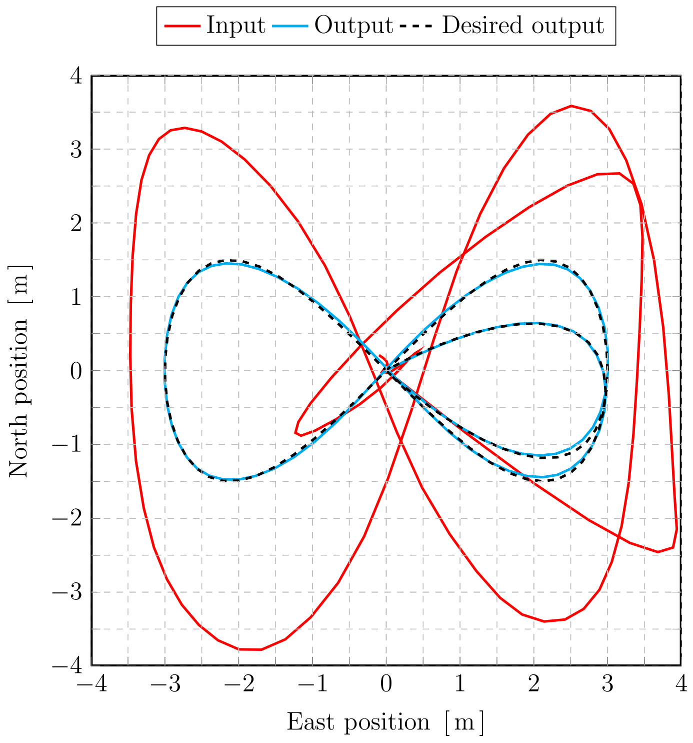


Figure 3.17: Converged iteration ($j = 25$) - 8-shape trajectories

The results of the virtual simulation in terms of the performance indicators, *i.e.* 'average' errors over the iterations is depicted in Figure 3.22.

Despite the large initial discrepancy, the drone learns rapidly how to precisely track the reference trajectory over the next few iterations. After the first iterations, the relative 'average' position errors decrease consistently as it can be seen by the adimentional and relative learning performance indicators displayed in Table 3.3.

In the simulation considered, after the fourteenth iteration, the trajectory can be considered to be learned. In fact, the performance parameter $\frac{e_{pos,j}}{e_{pos,0}}$ in percentage remains fixed to 1.9 % and the percentage relative error variations $(-3\% < \frac{e_{pos_j} - e_{pos_{j-1}}}{e_{pos_j}} \cdot 100 < 3\%)$ are comparable to non-repetitive variabilities of the system. In practical applications, to limit the number of trials, the trajectory can be considered to be learned after the first two or three iterations besides the reference one where most of the tracking error is reduced (as it will be done in the experiments conducted).

In order to provide insight into the computational cost associated to the simulation, the simulation times of the main MATLAB scripts are summarized in the Table 3.4. It is worth mentioning the time required by the calculator to compute the minimization problem (`fmincon`) which is about 10 s for each iteration for the simulation detailed.

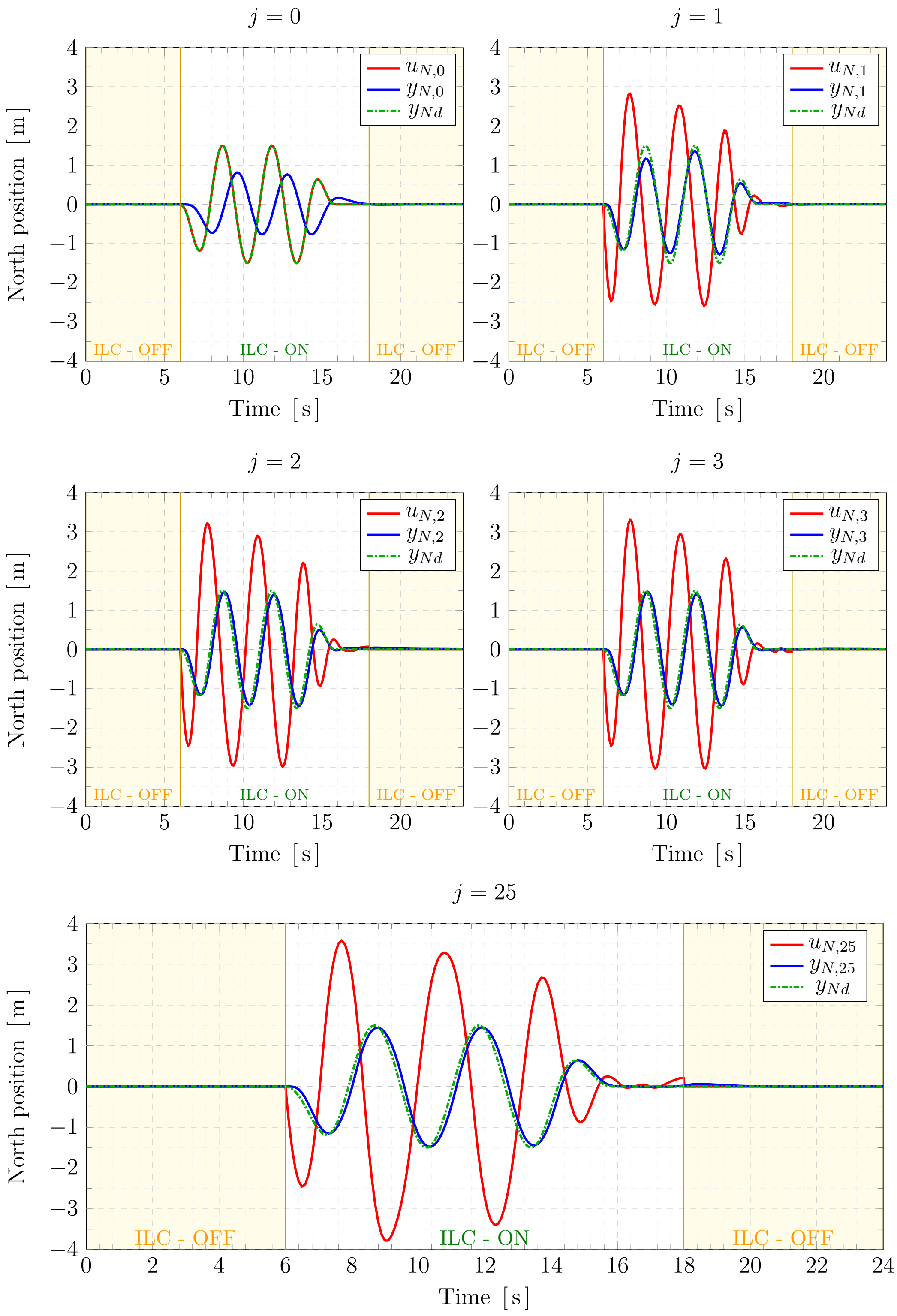


Figure 3.18: Quadrotor input, output and reference North-position over the iterations

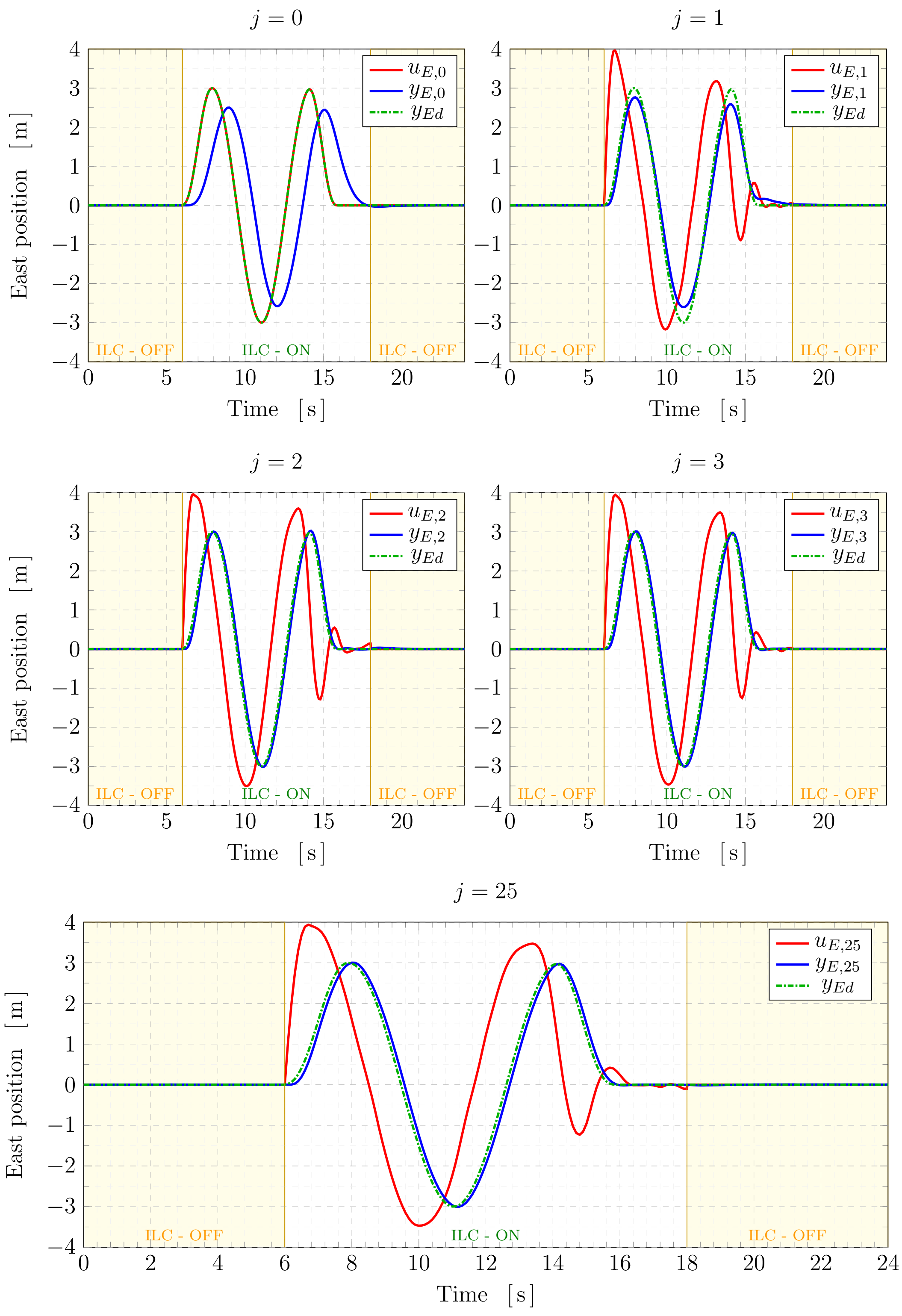


Figure 3.19: Quadrotor input, output and reference East-position over the iterations

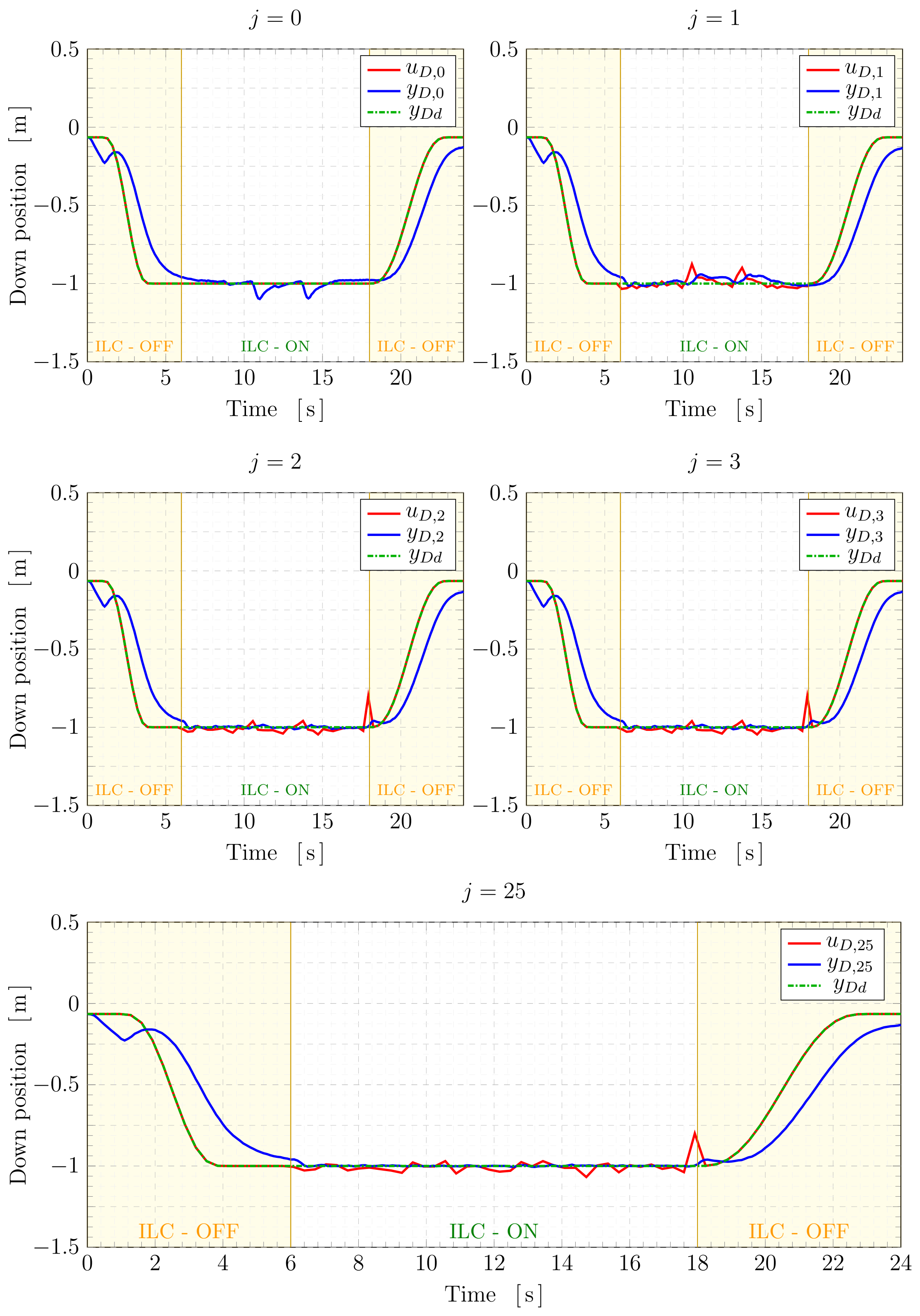


Figure 3.20: Quadrotor input, output and reference Down-position over the iterations

Iteration	$\frac{e_{pos_j}}{e_{pos_0}} \cdot 100$ [%]	$\frac{e_{pos_j} - e_{pos_{j-1}}}{e_{pos_j}} \cdot 100$ [%]
$j = 0$	100.00	-
$j = 1$	16.6	-83.5
$j = 2$	4.6	-72.4
$j = 3$	3.3	-28.5
$j = 4$	2.9	-9.6
$j = 5$	3.0	$+0.6$
$j = 6$	2.6	-11.4
$j = 7$	2.4	-10.1
$j = 8$	2.3	-3.8
$j = 9$	3.0	$+31.7$
$j = 10$	2.3	-22.8
$j = 11$	2.2	-4.9
$j = 12$	2.1	-5.0
$j = 13$	2.0	-6.6
$j = 14$	1.9	-1.8
$j = 15$	1.9	$+0.8$
$\vdots$	$\vdots$	$\vdots$
$j = 25$	1.9	-1.2

Table 3.3: Learning performance indicators

3.4 Main K-ILC design parameters choice

In this section, it is discussed how the different scalar K-ILC parameters are selected in order to improve the tracking performance of the quadcopter.

Optimization problem parameters The optimization problem is influenced mainly by the choice of two scalar parameters: w_e and $w_{\Delta u}$, or, to be more specific, by their relative weight. For this reason the weighting scalar term associated to the error is fixed, $w_e = 1$, while the weight linked to the input update (from an iteration to the next) is left free to vary. The minimization problem is also constrained by inequality and equality parameters.

$w_{\Delta u}$ and inequality constraint parameters (mainly the constraint in the maximum absolute value of the second derivative $\bar{\mathbf{u}}_{d2}$) are selected with the aim of reducing the jitter and smoothing the inputs.

The nonsmooth feedforward signals reduce the effectiveness of the learning as the higher noise that arises when driving a system by fast changing inputs reduces the repeatability of the experiment.

From the other side, the penalizing weight $w_{\Delta u}$ and the constraint upper lim-

Function Name	Calls	Total Time [s]	Self Time [s]	% Time[a]	Total Time Plot (dark band = self time)
● `main`	1	123.9	15.1		
- `run KILC`	1	55.1		44.5	
- `run Data & figure saving`	1	19.4		15.7	
- `run App`	1	17.5		14.1	
- `run ILC Plots`	1	11.7		9.5	
- `run Reference signals`	1	8.3		6.7	
- `others`	1	11.8		9.5	
● `KILC`	1	64.9	0.1		
- `run Launch Simulator`	1	55.1		84.8	
- `fmincon`	1	9.8		15.1	
- `others`	1	0.1		0.1	
● `Launch_Simulator`	2	63.0	1.5		

User choices: $\omega_{traj} = 1\,\text{rad/s}$, ILC-Mode: 1, Yaw following: '*on*', $N_j = 1$;
Design parameters: $\sigma = 0.1$, $\eta = 0.01$, $p_0 = 100$, $w_e = 1$, $w_{\Delta u} = 0.6$;
Additional parameters: $f_c = 250Hz$, $f_{ILC} = 20Hz$;
PC characteristics: Hp, Intel CPU i7-10510U (1.8GHz), 16 GB RAM, OS: Windows 10 (64-bit)

Table 3.4: Simulation times

[a]**Self time** is the time spent in a function excluding the time spent in its child functions.

its affect as well the learning performance since the ILC put more emphasis in smoothing the input rather than reducing the tracking error.

This effect is appreciable in Figure 3.23 which shows the quadrotor performance variations with $w_{\Delta u}$.

Kalman filter parameters The disturbance estimation determines the learning behaviour of the algorithm. The scalar parameter p_0 defining the initial variance of the disturbance predicted error matrix does not affect much the simulation. The noise model is characterized mainly by the scalar parameters σ_j and η defining respectively the process variance and the measurement variance matrices.

As discussed, both terms are chosen to constant over trials. Their ratio $\frac{\sigma}{\eta}$ determines how much the Kalman filter trust the process model with respect to the measurement data. A larger ratio implies a greater reliability in the measurement data.

Figure 3.24 illustrate the learning performance of the K-ILC quadrotor simulation for three different noise parameter ratios. As expected, a larger ratio $\frac{\sigma}{\eta}$ results in a faster convergence speed considering that the Kalman filter allows the disturbance estimate to change quickly.

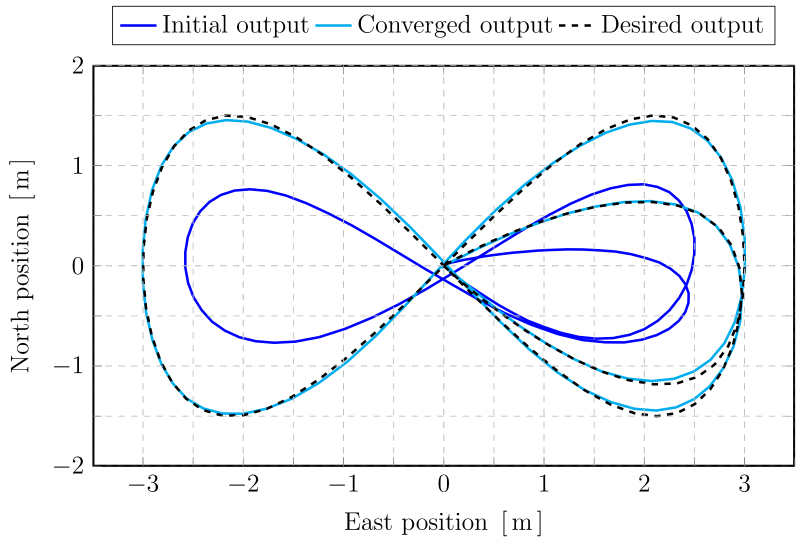


Figure 3.21: Initial ($j = 0$) and converged ($j = 25$) iterations - 8-shape trajectories

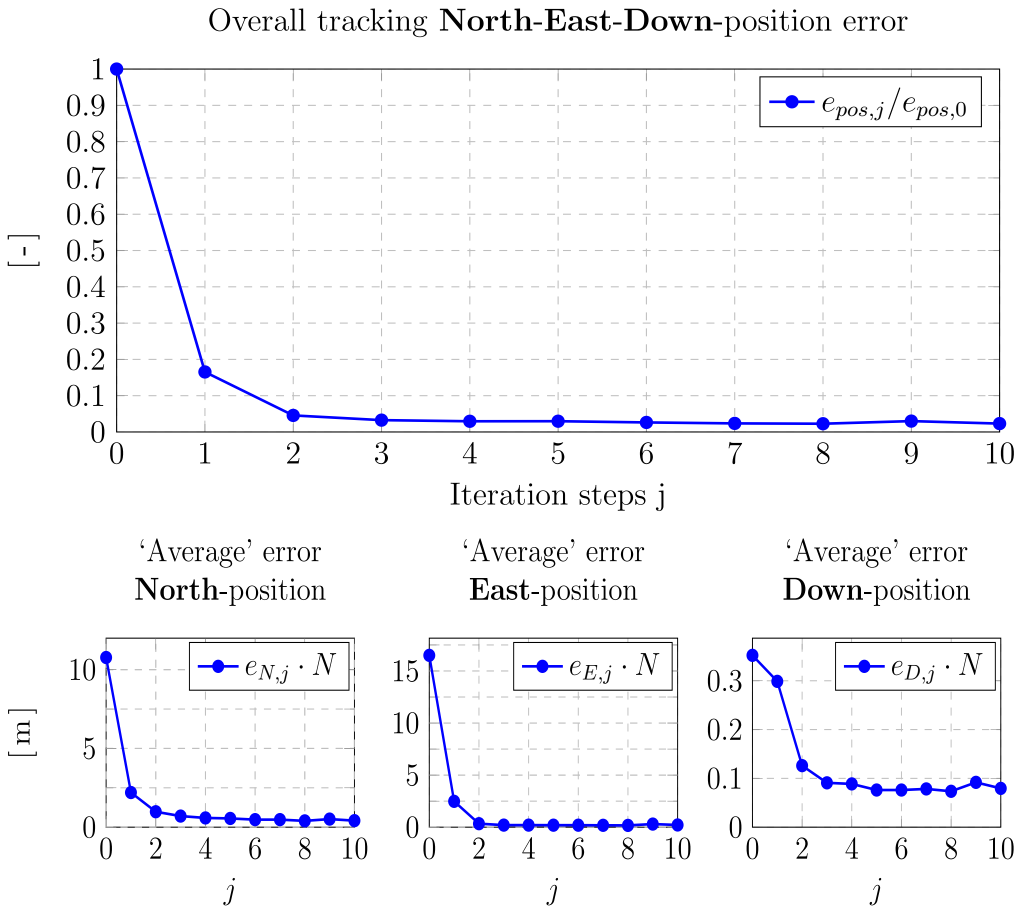


Figure 3.22: Simulation - learning performance analysis

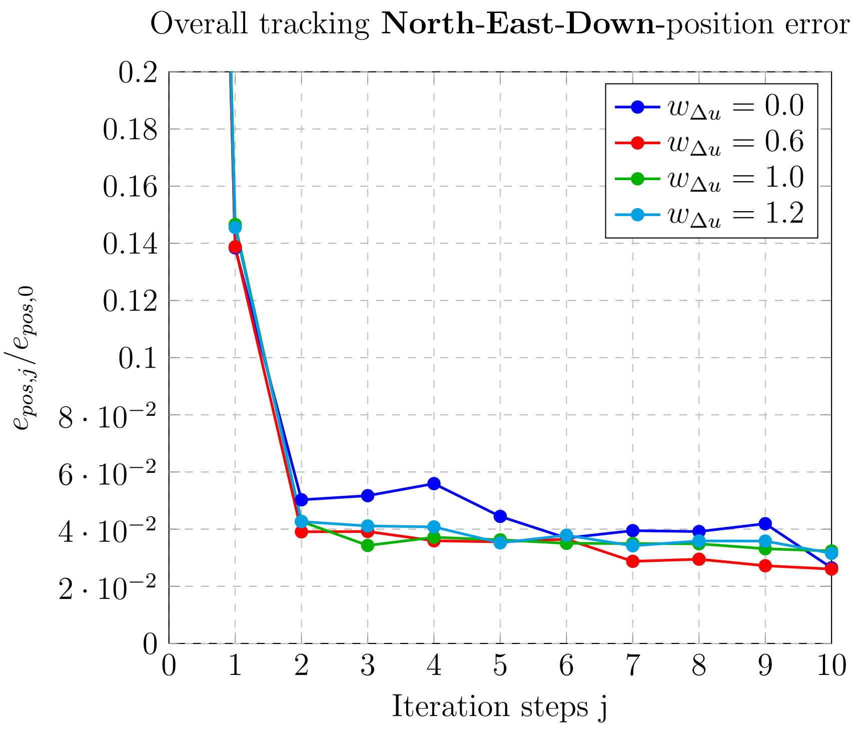


Figure 3.23: Learning performance for different input weight parameters $w_{\Delta u}$

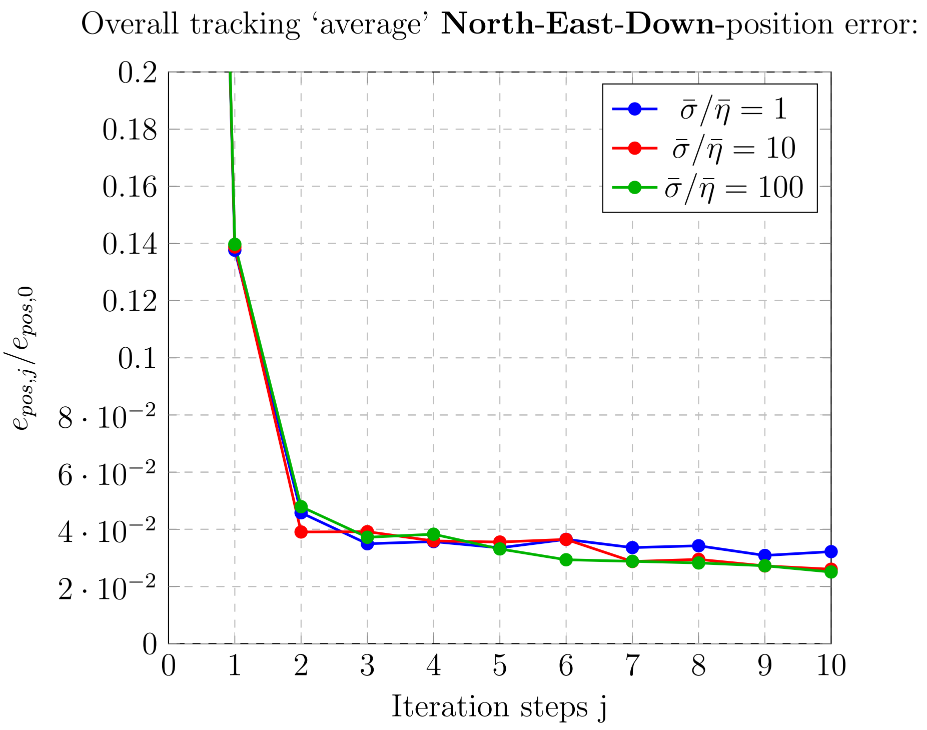


Figure 3.24: Learning performance for different noise parameter ratios $\bar{\sigma}/\bar{\eta}$

Chapter 4

Experimental setup, flight testing and results

In this chapter the trajectory tracking performance of the norm-optimal Kalman iterative learning control algorithm is tested in a real drone manoeuvre. First, an overview of the experimental setup (hardware and software) is illustrated. Then, the flight tests executed are described. Finally the experimental results are presented and discussed.

4.1 Experimental setup

This section describes the hardware and software components of the system architecture used for the flight tests.

4.1.1 Drone

The K-ILC has been implemented in the drone in Figure 4.1. This drone is a Micro Aerial Vehicles (MAV) fixed-pitch quadrotor, and, as such, it has a maximum take-off weight below 300 g, a flight autonomy above 10 minutes and a reduced geometry (the inner axis is lower than 200 mm).

It has 4 motors, 4 Electronic Speed Controls (ESCs), 4 propellers, a Lithium-Ion Polymer battery, a Flight Control Unit (FCU) and NanoPi NEO Air companion computer. The drone characteristics are reported in the Table 4.1. The drone FCU and Companion computers are detailed below.

4.1.2 Flight Control Unit

The Flight Control Unit mounted in the UAV regulates simultaneously the Revolutions Per Minute (RPM) of each motor to stabilize and control the drone. The FCU employed is the electronic board Pixhawk Mini [19] (produced by Holybro,

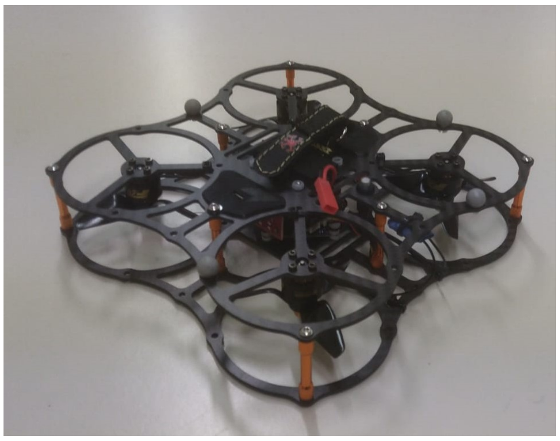


Figure 4.1: Drone

Weight	270 g
Dimensions	20 × 20 × 4 cm
Battery	LIPO 950 mAh 3S
Propellers	3 blades 3040
Motors	T-motor F20II KV3250

Table 4.1: Drone characteristics

Figure 4.2). It employs sensors,including 3-axes accelerometer, 3-axes gyroscope, magnetometer and barometer. Its main features are:

- Dimentions: 38 × 42 × 12 mm;
- Weight: 15.8 g
- Processor: STM32F427 based on 32 bit Cortex M4 core 168 MHz CPU and 256 kB SRAM.
- Interfaces: UART serial port for GPS, spektrum DSM/DSM2/DSM-X satellite compatible RC input, Futaba S BUS compatible RC input, PPM sum signal RC input, I2C for digital sensors, CAN for digital motor control with compatible controllers, ADC for analog sensors and micro USB port;

The MAVLink [20] (which stands for Micro Air Vehicle Link) is the protocol used for serial communication between the FCU and the companion computer, for communications with off-board APIs and with the Ground Control Station (GCS).

PX4 [21] is the open source autopilot firmware installed in the Pixhawk Mini. It executes low-level control algorithms (such as flight stabilization), within its dedicated micro-controller and exploiting information coming from high-rate inertial sensors.

QGroundcontrol is the software used for the configuration of the Pixhawk Mini FCU through its intuitive GUI. The software also provides full flight control and mission planning and it is used to monitor real-time data.

Figure 4.2: Pixhawk Mini FCU

4.1.3 Companion computer

The companion computer is the computer embedded in the drone that communicates with the FCU firmware PX4 using the MAVLink protocol. It is employed to implement high-level process that require a computational power which cannot be elaborated by the FCU.

The NanoPi NEO Air companion [22] (Figure 4.3) is installed in the drone. It receives the commands from the GCS and elaborates the information to the FCU. Its features are displayed in Table 4.3.

The Robot Operating System (ROS) [23] and MAVproxy are installed into the companion computer. The ROS package MAVROS enables MAVLink extendable communications between computers running ROS. MAVproxy: is the package intended for a minimalist portable and extendable Ground Control Station required to control the Pixhawk Mini FCU of the UAV from the GCS using the MAVLink protocol.

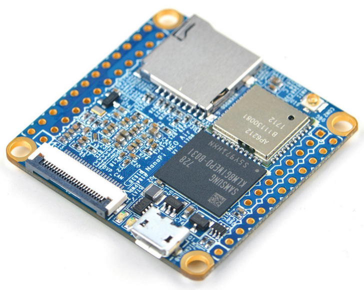


Figure 4.3: NanoPi NEO Air companion

CPU	Quad-core Cortex-A7 1.2 Ghz
RAM	512 MB
Wireless	2.4 GHz 802.11 b/g/n
Dimensions	40×40 mm
Weight	7.9 g
Power	5 V - 2 A

Table 4.2: NanoPi NEO Air features

4.1.4 Flying arena

The quadrotor tracking performance has been tested in the Flying Arena for Rotorcraft Technologies (FlyART) of Politecnico di Milano. The indoor facility is a dedicated testbed for motion control researches: it has flight volume of $6 \times 12 \times 4$ meters.

In the following the Motion Capture system (Mo-Cap) and of the Ground Control Station (GCS) are presented.

4.1.5 Motion Capture system

The FlyART is equipped with an high-precision Motion Capture system which consists of 16 Infra-Red (IR) sensitive OptiTrack cameras each of which is ringed by IR flood lights (Figure 4.4a). These cameras are calibrated in position and orientation so that the measurement subject is always detected by multiple cameras. The system provides accurate position and attitude information for any properly marked vehicle. Markers sensitive to infrared light are mounted on top of the drone as in Figure 4.4b. The tracking accuracy of the UAV depends on the frequency (cameras rate) with which data information are transmitted. This can be selected in a the range between 30 to 240 Hz.

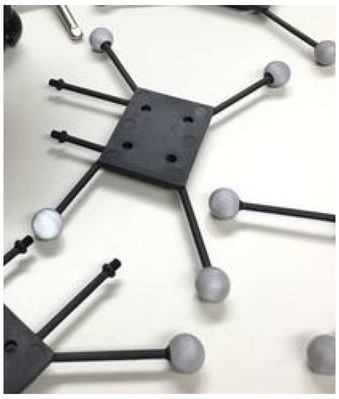
The Motion Capture system is controlled by the ground station PC using Motive [24]; the software allows the user to calibrate the system, it also provides interfaces for capturing and processing 3D data, that can be recorded or live-streamed.

4.1.6 Ground Control Station

The Ground Control Station is the UAV control centre. It consists of two PCs with different Operating Systems (OS): the Motive software is installed in a Windows 10 PC, whereas the Robot Operating System (ROS) and MATLAB run in Ubuntu Linux.

The GCS communicates with the UAV via wireless telemetry: it receives the UAV data, including the position and attitude measurements, by the Mo-Cap system at a frequency of 100 Hz and sends the control input signals to the quadrotor.

(a) IR camera



(b) IR markers

Figure 4.4: Motion Capture System

These setpoints generated by the GCS PC running the K-ILC MATLAB script are transmitted with a frequency rate of 20 Hz

4.2 Flight testing

The parameter selected for the flight test are reported in Table 4.3. The K-ILC design parameters displayed are the one previously selected during the different virtual simulations. For safety reasons, to prevent the drone from hitting the nets delimiting the flying arena the trajectory is shrunken reducing the scaling parameter from 3 to 1 ($s = 1$).

Exploiting the presented setup, first of all, an identification of the dynamics of the drone used in the experiment has been carried out considering the drone as a *black-box system* (*i.e.* a system which can be viewed in terms of its inputs and outputs, without any knowledge of its internal workings).

The aim is to build the LTI model of the system representing the dynamics of the drone in position which is required for the functioning of model-based ILC algorithm considered. In fact, the knowledge of the transfer functions linking the setpoint in position to the actual local position of the drone (in the NED reference frame) allows to defines the state matrices $\mathbf{A}_d$, $\mathbf{B}_d$ and $\mathbf{C}_d$ and to compute, in turns, the lifted form matrix $\mathbf{F}$ of the system.

Three separate experiments have been performed to validate the Kalman iterative learning control algorithm. The goal is to apply the algorithm to an highly manoeuvrable autonomous quadcopter for precisely tracking a predefined high-performance trajectory.

For a comparison with the simulation presented in the previous chapter, the manoeuvre to desired to be learnt is a fast eight-shape trajectory (which includes the two splines connecting the eight-figure to the initial hovering point).

In order to avoid major damages to the drone, the first experiment is executed with a speed parameter equal to half the desired value (*i.e.*, equal to $\omega_{traj} = 0.5$). Four trails has been executed considering a iterative controller acting in the three position variables (North-East-Down). The second flight test executed consists of three iterations performed with the same parameters.The ILC is applied in this case to the North and East position measures only. Once the effectiveness of controller has been validated for a low-speed manoeuvre, the last flight test, which includes three different trials, is performed at the desired speed ($\omega_{traj} = 1$) considering the same simplified ILC algorithm. In all three tests the equality constraints are removed in the optimization problem.

The three experiments conducted are summarized:

1. $\omega_{traj} = 0.5\,\text{rad/s}$, $N_j = 3$, 3D-trajectory, equality constraints removed;
2. $\omega_{traj} = 0.5\,\text{rad/s}$, $N_j = 2$, 2D-trajectory, equality constraints removed;
3. $\omega_{traj} = 1\,\text{rad/s}$, $N_j = 2$, 2D-trajectory, equality constraints removed;

where N_j is the number of iteration executed in addition to the reference. The flight mission panned for the set of experiments for each trial is summarized as follows:

- Take-off to an altitude of 1.5 m;
- Hovering at the centre of the cage till the drone is stabilized in position;
- Execution of the trajectory to be learned;
- Offline input update;

4.3 Experiment results

The first results obtained considering the 'slow' manoeuvre (with $\omega_{traj} = 0.5\,\text{rad/s}$ and with a maximum speed of about 0.82 m/s) shows that the ILC do not improve the flight performance. The reason of this is traced back to the non-negligible errors in the initial position recovery which contradicts Arimoto's postulate **P3** in 1.2. The mismatch in the initial conditions at the beginning of each iteration is mainly linked to the discrepancy in altitude. In fact, the ILC compensates for the initial tracking errors (mainly in altitude) dramatically at the beginning of the manoeuvre by corrupting the drone performance. In this way, the correcting ILC signals which act at the beginning of the trajectory are counterproductive and do not drive the UAV to improve the manoeuvre execution.

The difficulty in stabilizing the drone at the same position before the trial starts led to a problem simplification. Since the trajectory to be learned is bidimensional, the ILC algorithm is modeled considering the North and East position data only and without taking into account altitude information (Down position data). In particular, the ILC algorithm, fed by North and East position lifted vectors (containing the local positions of the drone during the trial with respect to the two axis of the NED reference frame), returns the input update in North and East position at each time instant for the next iteration.

For these reasons, the experiments mentioned in this section are uniquely computed setting the ILC to work with 2 variables only (North and East position). The results presented are referred to a trajectory similar to the one described in the previously chapter composed by two 5-th order splines connecting a eight-shape path.

Results for experiment n.2 ($\omega_{\mathbf{traj}} = \mathbf{0.5\,rad/s}$, $\mathbf{N_j = 2}$, 2D-trajectory)

In the first iteration, the nominal input (which is equal to the desired trajectory) is applied to the quadcopter. As depicted in Figure 4.5, the resulting drone eight-shape trajectory in blue is not coincident to the desired trajectory (black dashed line). Despite an important improvement in the second iteration ($j = 1$, green line), the third iteration ($j = 2$, red line) do not bring any advantages.

The evolution of the North and East position in time for the different iteration is illustrated in Figure 4.6.

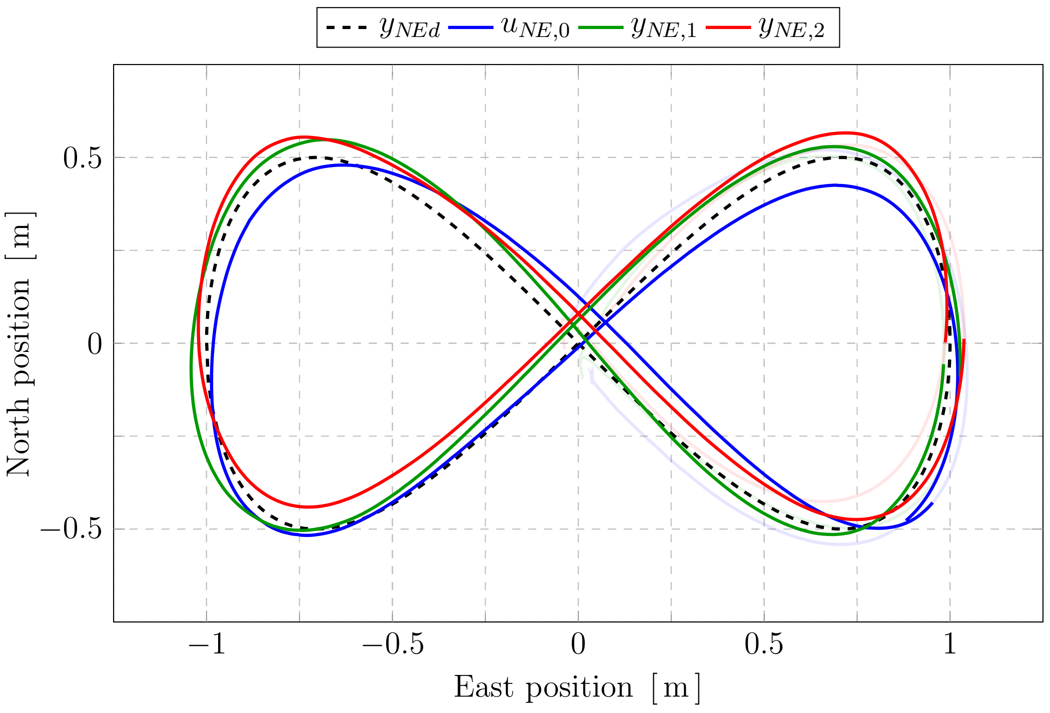


Figure 4.5: Experiment 2 ($\omega_{traj} = 0.5$) - 8-shape trajectories for iterations 0, 1, 2

The tracking performance results for the experiment n.2 are presented in Table 4.4. The aforementioned learning performance indicators are reported for the whole trajectory (which includes the connecting splines and the eight-shape path) and the eight-figure trajectory only. Note that the results are referred to the same experiment in which the ILC is active during the whole iteration.

The result of the third iteration of the second experiment are not convincing. Two are the possible explanations. A first reason is linked to the problem of stabilizing the drone in the same initial hovering position. As can be seen in Figure 4.6, the position of the drone at the beginning of the third trial (red line) is far off from the initial positions of the two previous trials both in North and East axes.

Another possible explanation is related to the optimization process. In fact, since the ILC working frequency is kept constant, the optimization problem complexity increases significantly as the length of the lifted vectors nearly double.

Results for experiment n.3 ($\omega_{\mathrm{traj}} = 1\,\mathrm{rad/s}$, $\mathrm{N_j} = 2$, 2D-trajectory)

The results obtained considering the desired trajectory tracking speed parameter are analysed in detail. Three are the iterations performed with $\omega_{traj} = 1\,\mathrm{rad/s}$ considering a maximum speed of about $1.48\,\mathrm{m/s}$. The experiment results in terms of North and East positions followed by the drone of the iteration 0, 1 and 2 are shown in Figure 4.7, 4.8, 4.9.

In the initial trial (Figure 4.7), the input trajectory (in violet) is equal to the desired trajectory (black dashed line). The actual drone trajectory (in blue) is improved with the second and third trials: the green and red trajectories in Figure 4.8 Figure 4.9 are closer to the desired path.

A comparison between the results obtained in the three trials is depicted. In Figure 4.10, the North and East positions in time, covering the period in which the ILC is active are plotted.

The trajectories followed by the drone in the different trials relatives to the eight-shape path only, instead, are displayed in Figure 4.11.

In both figures it is evident how the ILC improves the tracking performance by compensating in the first place the system's delays in the output response.

The results, in terms of the learning performance indicators based on the average error between the desired trajectory and the one actually followed by the drone, are summarized in Table 4.5.

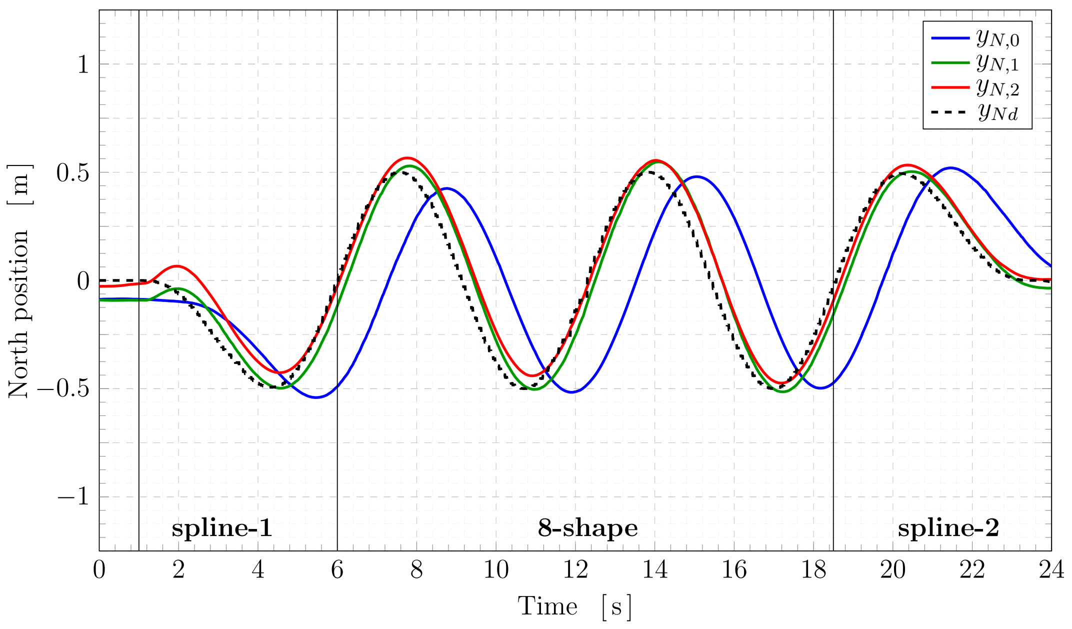
As can be seen from the error analysis, the K-ILC is very effective in improving the tracking performance of the UAV, being able to counteract the small initial errors in the positioning, the unmodeled system dynamics, process and measurement noises, achieving both low noise sensitivity and fast convergence. In the first two iterations the manoeuvre can already be considered learnt from a practical point of view as the 'average' relative error with respect to the reference iteration (trial $j = 0$) goes to $7.3\,\%$ ($3.8\,\%$ considering the eight-figure path only).

Symbol	Value	Description
ILC-Mode	1	Functioning mode of the K-ILC
Yaw following	‘OFF’	Yaw following enabler
s	1	Scaling trajectory coefficient
f_c	250 Hz	Feedback controller operating frequency
f_{ILC}	20 Hz	ILC controller operating frequency
f_{Mo-Cap}	100 Hz	Motion capture operating frequency
σ	0.1	Process noise scalar parameter
η	0.01	Measurement noise scalar parameter
p_0	100	Initial variance $\mathbf{P}_0$ scalar parameter
w_e	1	Next iteration tracking error scalar weight
$w_{\Delta u}$	0.6	Input change scalar weight
$\bar{u}_l$	4	Max/min input (absolute) constraint coefficient
$\bar{u}_{d1}$	25	Max/min input constraint coefficient of the 1st derivative
$\bar{u}_{d2}$	20	Max/min input constraint coefficient of the 2nd derivative

Table 4.3: Default parameters for the quadrotor experiments

Trajectory	Iteration	$\frac{e_{pos_j}}{e_{pos_0}} \cdot 100$ [%]	$\frac{e_{pos_j} - e_{pos_{j-1}}}{e_{pos_{j-1}}} \cdot 100$ [%]
8-shape + splines	$j = 0$	100.0	-
	$j = 1$	22.0	-78.4
	$j = 2$	23.7	$+7.7$
8-shape	$j = 0$	100.0	-
	$j = 1$	22.3	-77.7
	$j = 2$	20.6	-7.7

Table 4.4: Experiment 2 - learning performance results for $\omega_{traj} = 0.5\,\text{rad/s}$



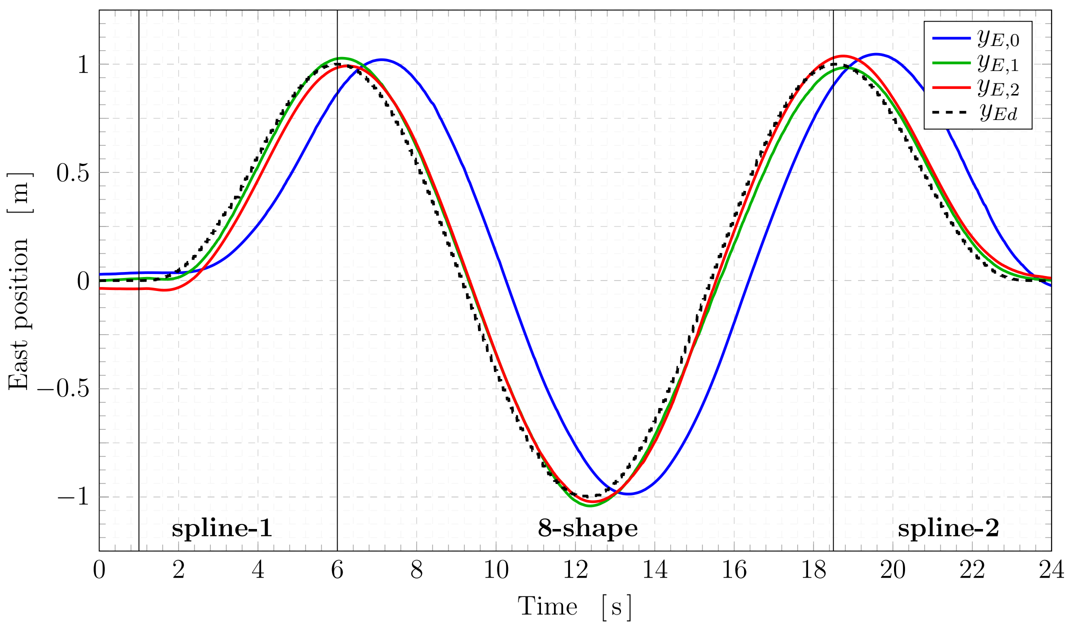


Figure 4.6: Experiment 2 ($\omega_{traj} = 0.5$) - North and East UAV positions for iterations 0, 1 and 2

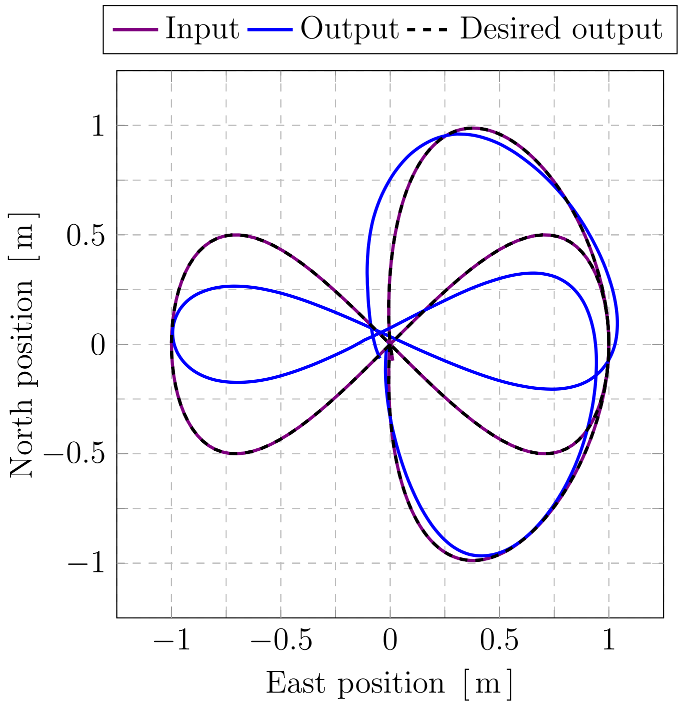


Figure 4.7: Experiment 3 ($\omega_{traj} = 1$) - input, output and desired trajectories for iteration $j = 0$

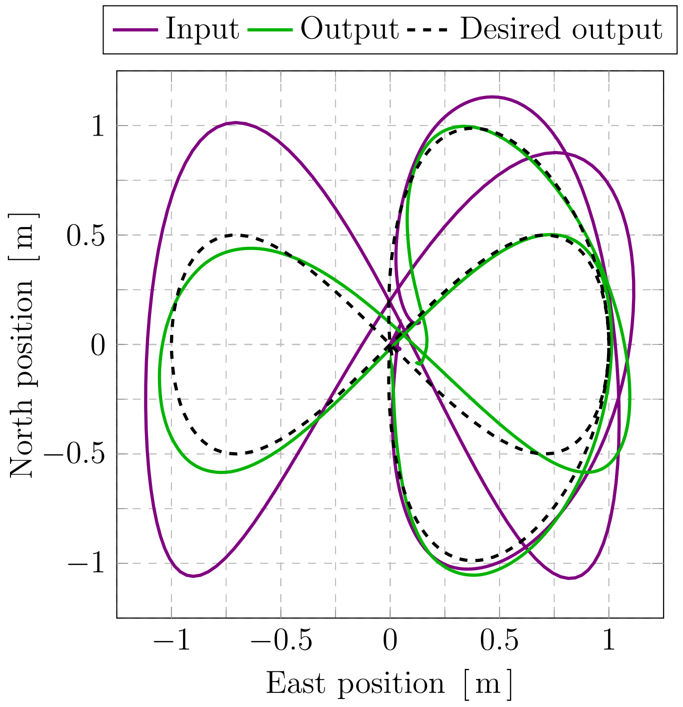


Figure 4.8: Experiment 3 ($\omega_{traj} = 1\,\text{rad/s}$) - input, output and desired trajectories for iteration $j = 1$

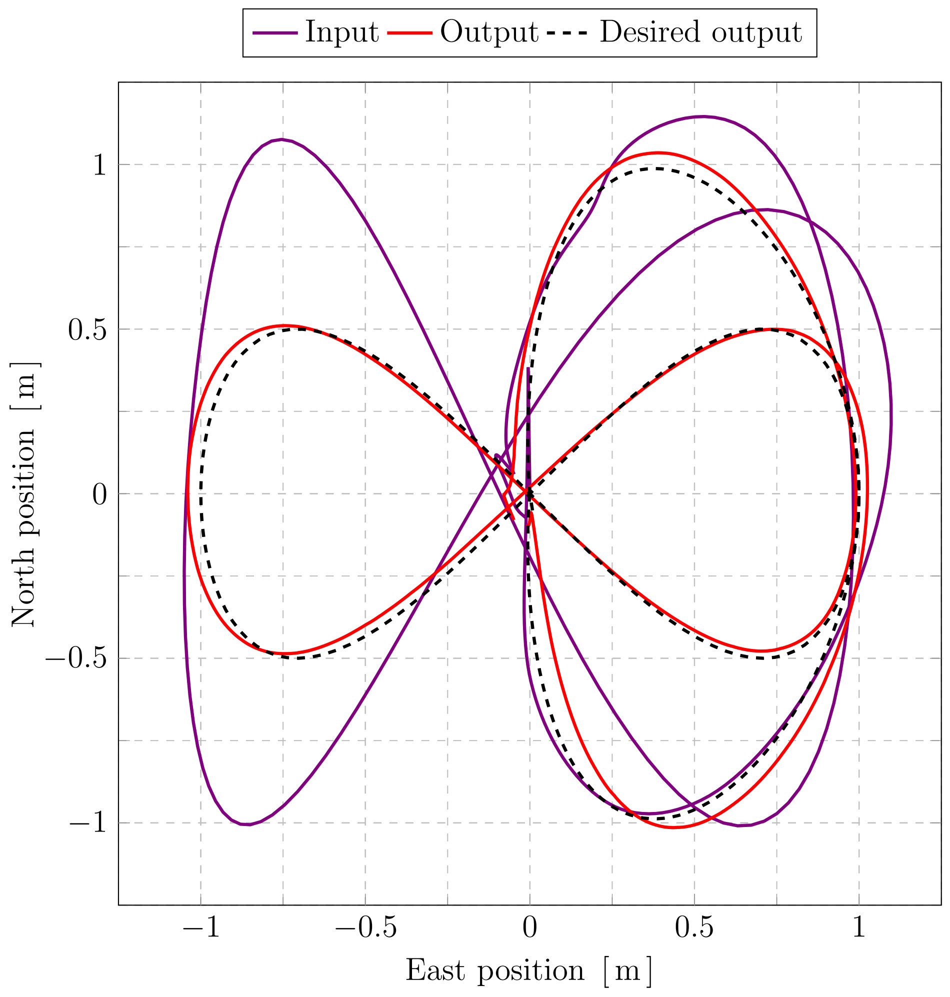
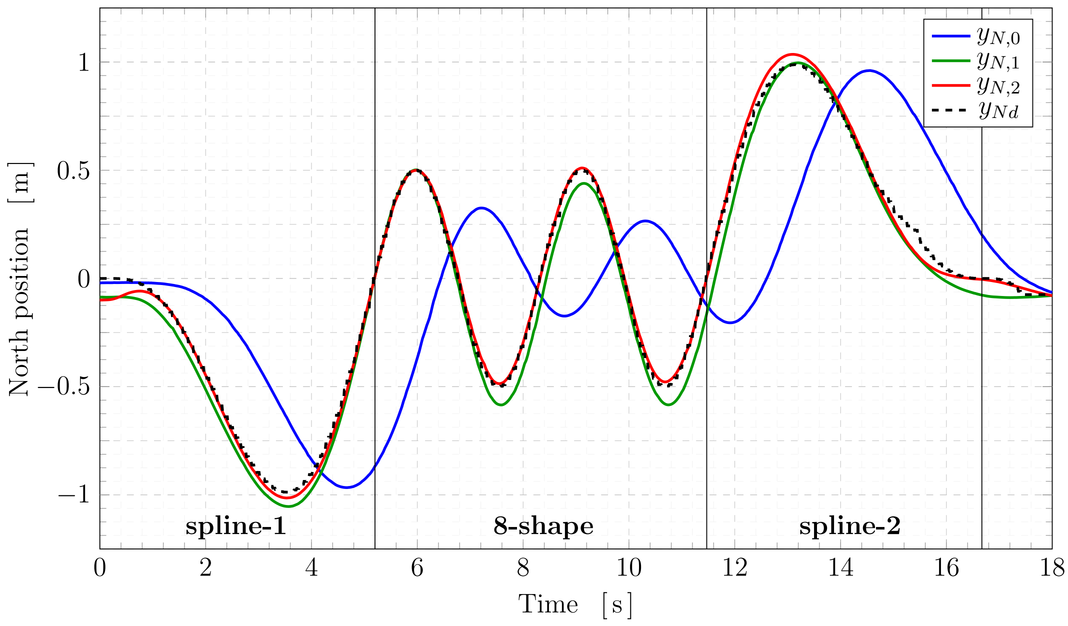


Figure 4.9: Experiment 3 ($\omega_{traj} = 1\,\mathrm{rad/s}$) - input, output and desired trajectories for iteration $j = 2$

Trajectory	Iteration	$\frac{e_{pos_j}}{e_{pos_0}} \cdot 100$ [%]	$\frac{e_{pos_j} - e_{pos_{j-1}}}{e_{pos_{j-1}}} \cdot 100$ [%]
8-shape + splines	$j = 0$	100.0	-
	$j = 1$	14.8	-85.2
	$j = 2$	7.3	-50.9
8-shape	$j = 0$	100.0	-
	$j = 1$	11.3	-88.7
	$j = 2$	3.8	-66.4

Table 4.5: Experiment 3 - learning performance results for $\omega_{traj} = 1$



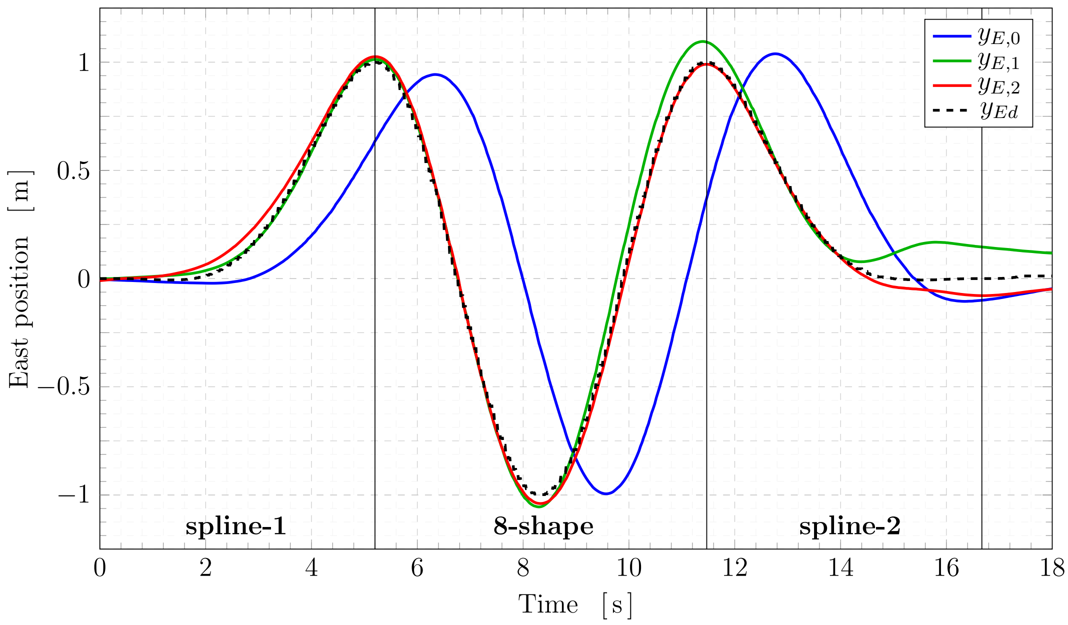


Figure 4.10: Experiment 3 ($\omega_{traj} = 1\,\text{rad/s}$) - North and East UAV positions for iterations 0, 1 and 2

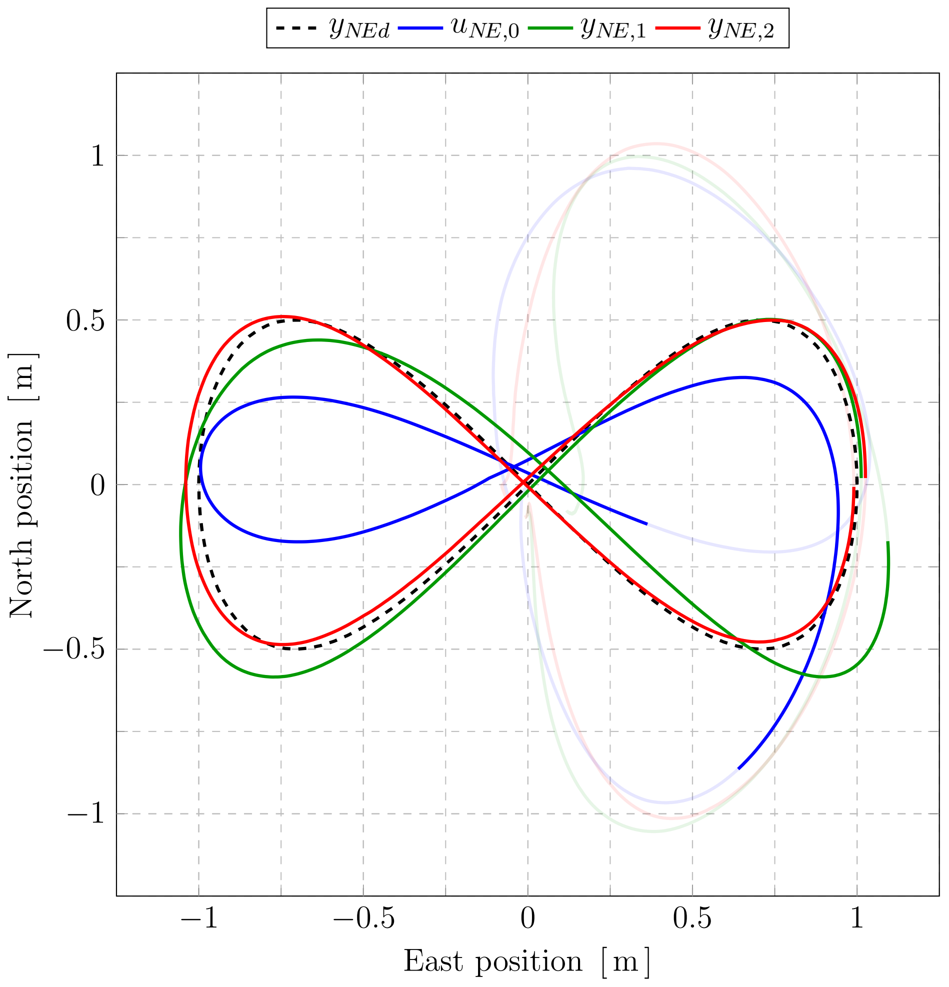


Figure 4.11: Experiment results for $\omega_{traj} = 1\,\mathrm{rad/s}$: 8-shape trajectories for the iterations 0, 1, 2

Conclusions

The purpose of this thesis has been to design, simulate and experimentally validate a model-based, norm-optimal, Kalman-enhanced iterative learning control algorithm for precise UAV trajectory tracking. In particular, the iterative controller has has been implemented in a high manoeuvrable autonomous quadrotor vehicle to improve the execution of a complex manoeuvre.

The estimator has been designed and integrated in the typical optimization-based architecture so as to estimate repetitive disturbances happening during the different trials. This information has been exploited to shape a new input signal to be used in the next iteration for rejecting the iteration-repetitive errors. The neat separation of disturbance estimation and input update steps of the K-ILC has provided an intuitive design setup, making easier the choice of the parameters regulating the learning process and the system performance. In this work, the model-based ILC for trajectory tracking has been applied to a pre-existing quadcopter model considering a 'black-box approach'.

The main results are summarized in the following:

- the virtual simulations show that the K-ILC algorithm can achieve fast convergence and good noise rejection by adjusting the iteration-varying Kalman gain to improve the repetitive disturbance estimation and by adapting the input signal optimally for every trial. The non-causal inputs, in facts, allow the quadrotor model to improve its performance in tracking a predefined path over the iterations by compensating for repetitive errors. In this way, a noise level comparable to the level of non-repetitive noise is achieved.

- The ILC system has proved to be robust against disturbances resulting mainly from unmodeled dynamics, process, measurement and environmental noises. The simulations shows that the K-ILC works well even when a rough approximation of the dynamics of the system under study is considered. In addition, the iterative controller is also robust to the design parameters. The choice of the learning parameters certainly influences the way the system performs, but do not prevent the system to improve its tracking performance.

- The approach was successfully applied to a drone. The experiments presented in the previous chapter demonstrated the effectiveness of the K-ILC algorithm in tracking a bidimensional trajectory by properly compensate for the system's delays and model errors. The tracking error is reduced drastically right after the first iteration. The resulting performance, hoverer, relies on the quality of the estimate: non-repetitive noise can be compensated by the quadcopter feedback controller only, as the iteration-varying input is not effective. This is why the accuracy of the tracking is limited by the level of non-repetitive noise and is highly influenced by relevant discrepancies in the initial conditions; thus, achieving an accurate initial state is crucial.

Starting from the results obtained the recommendations and possible future developments are:

- the UAV needs to regain the same initial position at every trial in order for the iterative algorithm to perform optimally. This could be achieved introducing a start condition which would enable the drone to start the iteration only when the initial state lies within specified bounds.
- To avoid problem in solving the optimization step for the input update, the ILC working frequency could be adapted on the basis of the duration of the trials so as to shorten the length of the lifted vectors (*e.g.* when the manoeuvre to learn is 'slow'). Considering the the initial state bounds and an the frequency adjustment, the first experiment can be executed again.
- the K-ILC can be tested in three-dimensional trajectories. This could be done considering more effective methods for solving the optimization problem as the size of the variables involved increases. This is not an issue in terms of the computational time required because of the 'offline' nature of the iterative control, as the computation is not done in real time.
- A future research problem is to investigate the capability of the quadrotor in performing in outdoor scenarios, *e.g.*, in presence of wind gusts and variable initial conditions. In practice, the K-ILC scheme should be implemented in quadrotors that have access to position information, obtained, for example, from GPS measurements.

Bibliography

[1] S. Lupashin and R. D'Andrea. Adaptive fast open-loop maneuvers for quadrocopters. *Autonomous Robots*, 33(1):89–102, 08 2012.

[2] Markus Hehn and Raffaello D'Andrea. A frequency domain iterative learning algorithm for high-performance, periodic quadrocopter maneuvers. *Mechatronics*, 24(8):954 – 965, 2014.

[3] Oliver Purwin and R. D'Andrea. Performing aggressive maneuvers using iterative learning control. pages 1731 – 1736, 06 2009.

[4] R. D'Andrea A. Schoellig, F. Mueller. Optimization-based iterative learning for precise quadrocopter trajectory tracking. *Autonomous Robots*, 33, 08 2012.

[5] F. L. Mueller, A. P. Schoellig, and R. D'Andrea. Iterative learning of feed-forward corrections for high-performance tracking. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3276–3281, 2012.

[6] N. Degen and A. P. Schoellig. Design of norm-optimal iterative learning controllers: The effect of an iteration-domain kalman filter for disturbance estimation. In *53rd IEEE Conference on Decision and Control*, pages 3590–3596, 2014.

[7] M. Tharayil D.A. Bristow and A.G. Alleyne. A survey of iterative learning. *Control Systems, IEEE*, 26:96 – 114, 07 2006.

[8] F. Miyazaki S. Arimoto, S. Kawamura. Iterative learning control for robot systems. In *Proceedings of Annual Conference of the IEEE Industrial Electronics Socitey, IECON*, Tokyo, Japan, October 1984a.

[9] J. Craig. A learning procedure for the control of movements of robotic manipulators. In *Proceedings of IASTED Symposium on Robotics and Automation*, page 108–111, San Francisco, CA, USA, May 1984.

[10] G. Casalino, G. Bartolini. In *Proceedings of American Control Conference*, page 1566–1572, San Diego, CA, USA, June 1984.

[11] S. Arimoto. *A Brief History of Iterative Learning Control*, pages 3–7. Springer US, Boston, MA, 1998.

[12] J. Wallén. Estimation-based iterative learning control. 09 2012.

[13] D. De Roover and O. H. Bosgra. Synthesis of robust multivariable iterative learning controllers with application to a wafer stage motion system. *International Journal of Control*, 73(10):968–979, 2000.

[14] J. Swevers T. Duy Son, G. Pipeleers. Multi-objective iterative learning control using convex optimization. *European Journal of Control*, 09 2016.

[15] T. D. Son, G. Pipeleers, and J. Swevers. Robust monotonic convergent iterative learning control. *IEEE Transactions on Automatic Control*, 61(4):1063–1068, 2016.

[16] A. Schöllig and R. D'Andrea. Optimization-based iterative learning control for trajectory tracking. In *2009 European Control Conference (ECC)*, pages 1505–1510, 2009.

[17] Flyart-quadrotor simulator. `https://gitlab.com/flyart/quadrotor-simulator`.

[18] M. Giurato. Design, integration and control of a multirotor UAV platform. Master's thesis, Politecnico di Milano, 2015.

[19] Pixfalcon flight controller. `https://docs.px4.io/v1.9.0/en/flight_controller/pixfalcon.html`.

[20] Mavlink. `https://mavlink.io/en/`.

[21] Px4. `https://px4.io/`.

[22] Nanopi neo air. `http://wiki.friendlyarm.com/wiki/index.php/NanoPi_NEO_Air`.

[23] Robot operating system (ros). `https://www.ros.org/`.

[24] Optitrack - motive. `https://optitrack.com/products/motive/`.