UNIVERSITÀ DEGLI STUDI ROMA TRE

Dipartimento di Ingegneria
Corso di Laurea in Ingegneria Informatica

Tesi Di Laurea

Implementazione di un Controllore ad apprendimento iterativo con perdita di dati

Laureando

Caponi Marco

Matricola: 508773

Relatrice

Prof. Chiara Foglietta

Anno Accademico 2019/2020

Indice

Introduzione

L'apprendimento è una skill basilare dell'essere umano, serve per sopravvivere negli ambienti ostili durante l'intero arco di una vita.
Attraverso il processo di apprendimento possiamo migliorarci utilizzando le esperienze e conoscenze pregresse.
Il mio studio effettuato durante il periodo di tirocinio si basa proprio su questo meccanismo di apprendimento. Ad oggi, siamo arrivati ad un livello di tecnologia tale da poter far apprendere queste ***Basic skill*** a dei controllori in processi automatici.
In questo modo, introduciamo un nuovo sviluppo e metodo di controllo dell'apprendimento denominato ***ILC***: Iterative Learning control.
Il controllore ad apprendimento iterativo (ILC) è stato sviluppato e migliorato nel corso di tre decenni da quando è stato proposto, in quanto è una metodologia di controllo semplice ed efficace per sistemi ripetitivi applicata, per esempio, nei bracci robotici industriali oppure nei processi chimici che richiedono un elevato tasso di *precisione.*
Per approcciarsi a tutto ciò, ho sviluppato un ILC in modo tale da passare in ingresso un segnale di riferimento affetto da un rumore o disturbo (di tipo randomico) e attraverso un algoritmo che vedremo in seguito, il controllore riuscirà a recuperare e ricreare un segnale d'uscita similare al segnale passato come riferimento.
Per questo tipo di sistemi, le informazioni di tracciamento delle iterazioni precedenti sono essenziali e vengono utilizzate per generare il *comando di controllo* dell'iterazione successiva, in modo tale che le suddette prestazioni saranno gradualmente migliorate per raggiungere il nostro segnale finale.
Il sistema deve ripetere alcune attività di monitoraggio in un intervallo di tempo finito e prestabilito; l'input della successiva iterazione verrà generato in funzione degli input e degli errori di tracciamento delle iterazioni precedenti.
Per renderlo il più fruibile possibile, l'ILC è una strategia di controllo basata sull'iterazione, differisce dalle altre metodologie di controllo come, per esempio, il ***controllo del feedback*** oppure il **controllo *adattivo***, che si concentrano principalmente sull'evoluzione basata lungo l'asse temporale. Con l'ILC le prestazioni di tracciamento precise vengono raggiunte gradualmente lungo l'asse di iterazione imitando in questo modo le capacità di apprendimento dell'essere umano (basando le sue capacità e conoscenze a eventi passati).
L'ILC è progettato per sistemi che sono capaci di completare determinati task in un intervallo predefinito ed ha le seguenti caratteristiche:

1. Il sistema deve finire il suo task in un intervallo di tempo finito.
2. Il sistema può essere resettato ad un valore iniziale.
3. L'obiettivo di tracciamento è a iterazione invariante.

Il punto 1 è un'operazione che trova l'applicazione in molti ambienti automatici come una linea di produzione formata da molteplici robot che devono ripetere la stessa procedura in intervalli di tempo scanditi, prendiamo in esempio un robot che prende e posa uno specifico oggetto in diverse linee di assemblaggio. Definire e scandire i tempi per ogni operazione costituisce un elemento fondamentale per il corretto funzionamento dell'intero sistema.
Il reset del sistema definito nel punto 2, fa parte di un componente ben specifico dell'ILC, è uno dei punti più critici per un'applicazione pratica ed è uno dei punti cardini alla base della creazione di un processo automatico.
L'ultimo ma non per importanza, l'obiettivo di tracciamento è fondamentale per la ripetitività del sistema stesso, poiché, come l'uomo applica le sue conoscenze pregresse su nuovi obiettivi talvolta simili, ci aspettiamo che un sistema ILC reagisca in maniera similare su determinati obiettivi o task, migliorando e ottimizzando il suo processo.
Per concludere, la domanda più importante che ci poniamo è "Perché utilizzare ILC?"
Il vantaggio principale di ILC riguarda la progettazione delle *leggi di controllo* per la gestione dell'intero sistema che richiedono unicamente le conoscenze di base dei riferimenti di tracciamento e i segnali di input e output.
In altre parole, sono richieste poche informazioni riguardante il *plant (*chiamato anche impianto) e i sistemi matriciali che ne derivano, possono essere completamente sconosciuti avendo così un algoritmo di controllo semplice, intuitivo ed efficace.

Capitolo 1: Cos'è un ILC

Per iniziare a capire cos'è effettivamente un controllore ad apprendimento iterativo, possiamo immaginarci un impianto formato da molteplici robot allineati lungo un nastro trasportatore impiegati in uno stabilimento di produzione, eseguendo la medesima operazione più e più volte.
Tuttavia, se riuscissimo a migliorare le prestazioni di ogni singolo robot, l'impianto potrebbe funzionare ad una velocità maggiore il che avrebbe effetti positivi sulla produttività generale dell'intero sistema. Per far ciò ci viene in aiuto proprio il controllore ad apprendimento iterativo, un metodo per sistemi di controllo per funzionamenti di tipo ripetitivo che vedremo nel dettaglio in questa tesi.
Senza scendere troppo nel dettaglio, ILC si occupa di un unico compito di controllo; Di conseguenza tutte le informazioni vengono codificate e aggiornate in relazione ad un singolo parametro: *Il tempo.*

1.1 Il problema di ILC

> *"The real problem is not whether machines think but whether men do"*
> -B.F. Skinner

Abbiamo appreso quindi, che ILC mira a migliorare le prestazioni di un sistema con mezzi di ripetizione, ma non sappiamo ancora come funzionano e come sono fatti.
Questo ci porta nel cuore di ricerca di sistemi ILC, nel particolare, la costruzione e la successiva analisi di algoritmi.
ILC parte da una descrizione qualitativa di un comportamento di apprendimento il cui problema principale è quello di trovare un algoritmo che lo implementi.
Poiché questo non è affatto un problema facile, cerchiamo di delineare quali caratteristiche dovrebbe avere un corretto algoritmo.
Facciamo prima un passo indietro e chiediamoci da cosa scaturisce il bisogno di avere un controllo iterativo.
Parte di questa esigenza diventa importante quando ci concentriamo su situazioni in cui non è possibile avere un controllo convenzionale (ad esempio un *controllo in feedback*).
Grazie ad ILC, l'uso di diverse iterazioni lungo l'asse temporale, ci dà la possibilità di migliorare le prestazioni. È proprio per questo che la vera difficoltà è convertire questa possibilità in una realtà concreta.

1.2 Terminologia di base

In questa sezione si analizza il concetto di ILC in maniera più approfondita, il suo funzionamento e le proprietà che lo caratterizzano.
Il concept principale e generale di un ILC è raffigurato nella figura 1.1; la variabile y_d denota la *traiettoria di riferimento* (chiamata anche **Reference trajectory**). Alla k-esima iterazione, l'input u_k è alimentato dal plant che corrisponde proprio all'output del sistema y_k.
Vediamo infine co me la differenza tra l'output e l'input rappresentato dalla reference trajectory equivale all' *errore di tracciamento* (chiamato anche **Traking error**).

$$e_k = y_d - y_k \qquad (1)$$

Finché e_k sarà diverso da 0, implicherà che l'input u_k non è ottimo e dovrà essere migliorato nell'iterazione successiva k+1.
Questo specifico miglioramento è una funzione di combinazioni lineari formati da u_k ed e_k, nello specifico l'input $u_k + 1$ per la k+1esima iterazione verrà generato prima dell'esecuzione della k+1esima iterazione e subito dopo verrà inviato al *plant* per l'iterazione successiva.
Come risultato finale avremo un feedback a circuito chiuso formato lungo l'asse di iterazione.
Per riportare un esempio, se confrontiamo l'ILC con la nostra vita quotidiana, troviamo che le informazioni precedenti di input e output del plant corrispondono alle esperienze accumulate nell'arco della nostra vita. Le persone generalmente attuano una *strategia* per risolvere e affrontare un obiettivo basandosi sulle esperienze pregresse. Questa strategia è proprio il segnale input dell'ILC.

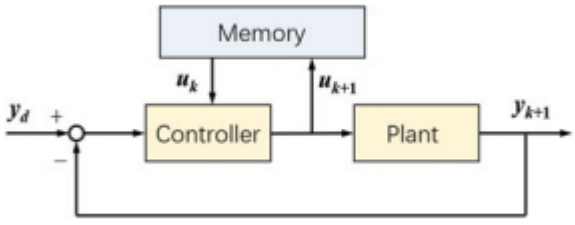

Figura 1.1 Struttura di un ILC

È importante far notare che l'ILC aggiusta il controllo lungo l'asse iterativo anziché lungo l'asse temporale. Questa è la differenza più importante con altri tipi di controllori come il **PID** (ampiamente utilizzati nei feedback di controllo).

D'altronde per sistemi ripetitivi, il *PID* genererà lo stesso *tracking error* per tutte le iterazioni mentre l'ILC lo ridurrà di iterazione in iterazione.

1.2.1 Formulazione base di un ILC

Illustriamo ora una formulazione base e rigorosa di un controllore ad apprendimento iterativo.
Per iniziare presentiamo il caso di un sistema a tempo discreto e più in avanti parleremo del sistema tempo continuo.

1.2.2 Il modello a tempo discreto

Consideriamo il seguente sistema a tempo discreto - invariante lineare:

$$x_k(t+1) = Ax_k(t) + Bu_k(t) \qquad (1.1)$$
$$y_k(t) = Cx_k(t)$$

Dove $x \in R^n$, $u \in R^p$ e $y \in R^q$ denotano rispettivamente lo stato del sistema, l'input e l'output. Le matrici A, B, C sono sistemi matriciali di appropriate dimensioni, *t* denota un istante di tempo arbitrario in un'operazione di iterazione, *t* = 0,1,2, . . ., N dove N è la lunghezza delle operazioni di iterazione.
Per semplicità diremo che $t \in [0, N]$ e k = 0, 1, 2 . . . indicherà l'esatta iterazione in cui ci troviamo.
Finché è richiesta una ripetizione di uno specifico task, lo stato iniziale dovrà essere resettato per ogni iterazione.

$$x_k(0) = x_0, \forall k \qquad (1.2)$$

La formula vista nel punto 1.2 è la condizione base di reset chiamata anche I.I.C (**identical initialization condition)**.
La traiettoria di riferimento che utilizzeremo per l'input è denotata da $y_d(t), t \in [0, N]$ con la condizione I.I.C che richiederà: $y_d(0) = y_0 \triangleq Cx_0$
Lo scopo di controllo dell'ILC è quello di progettare una *legge di aggiornamento* adeguata (denominata **Update law**) per l'input $u_k(t)$, in modo tale che il corrispondente output $y_k(t)$ possa tracciare $y_d(t)$ il più fedelmente possibile.
L'errore di traiettoria per sistemi discreti si definisce come:

$$e_k(t) = y_d(t) - y_k(t) \qquad (1.3)$$

1.2.3 Legge di aggiornamento

La legge di aggiornamento è invece una funzione di $u_k(t)$ ed $e_k(t)$ che genererà $u_{(k+1)}(t)$ la cui forma generale è la seguente:

$$u_{(k+1)}(t) = h[u_k(\bullet), .., u_0(\bullet), e_k(\bullet), .., e_0(\bullet)] \quad (1.4)$$

Quando la seguente relazione dipenderà solamente dall'ultima iterazione, la legge di aggiornamento prenderà il nome di ***first order ILC update law.*** Nel caso contrario, quando non dipenderà unicamente dall'ultima iterazione, prenderà il nome di ***high order ILC update law***. Generalmente per tenere l'algoritmo il più semplice possibile, si utilizzano leggi di aggiornamento di tipo *First order* come per esempio:

$$u_{(k+1)}(t) = h[u_k(\bullet), e_k(\bullet)] \quad (1.5)$$

L'update law più generale possibile viene scritta nel seguente modo ed è quella che ho utilizzato per progettare l'intero sistema ILC:

$$u_{(k+1)}(t) = u_k(t) + Ke_k(t+1) \quad (1.6)$$

P-type ILC update law

Definiamo K, la matrice di *guadagno d'apprendimento,* $u_k(t)$ è il nostro input della corrente iterazione mentre $Ke_k(t+1)$ è il termine di innovazione (denominato anche (*Innovation Term*).
Se il termine di innovazione è rimpiazzato da: $K[e_k(t+1) - e_k(t)]$ la legge di aggiornamento sarà:

$$u_{(k+1)}(t) = u_k(t) + K[e_k(t+1) - e_k(t)] \qquad (1.7)$$

D-*type ILC update law*

Per il sistema 1.1 e la legge di aggiornamento 1.6, una condizione base è che K soddisfi la seguente disuguaglianza

$$|| I - CBK || < 1 \qquad (1.8)$$

In questo modo vediamo come K non dipende dalla matrice A ma richiederà informazioni unicamente dalle matrici C e B.
Questo fatto ci fa notare un vantaggio dell'ILC dal punto di vista per cui questo tipo di controllore ha poca dipendenza dalle informazioni del sistema. In questo modo l'ILC può gestire quei problemi dovuti al tracciamento che hanno più incertezza senza troppi problemi.

1.3 Sistema a tempo continuo

Consideriamo ora il seguente sistema lineare a tempo continuo:

$$\dot{x}_k(t) = Ax_k(t) + Bu_k(t) \quad (1.9)$$
$$y_k(t) = Cx_k(t)$$

Il compito del controllo è di studiare y_k e tracciare i riferimenti desiderati y_d in un intervallo di tempo fisso $t \in [0, T]$ man mano che il numero di iterazioni k aumenta.
Uno schema ILC largamente utilizzato per sistemi continui è denominato *schema tipo Arimoto* ed è del tipo:

$$u_{(k+1)} = u_k + \Gamma \dot{e}_k \qquad (1.10)$$

Come per il sistema a tempo discreto, l'errore di traiettoria è dato dalla differenza del segnale d'ingresso e d'uscita: $e_k(t) = y_d(t) - y_k(t)$ mentre Γ è la diagonale della matrice di guadagno di apprendimento; Se Γ soddisfa:

$$|| I - CB\Gamma || < 1 \qquad (1.11)$$

Allora lo scopo del controllo è assicurato, se per esempio abbiamo:

$$\lim_{(k\to\infty)} = y_k(t) \to y_d(t) \quad (1.12)$$

Ovvero, l'uscita $y_d(t)$ sarà proprio la traiettoria di riferimento passata in ingresso $y_k(t)$.
Come per il sistema a tempo discreto la matrice di guadagno d'apprendimento non richiede nessuna informazione per quanto riguarda la matrice A. Ciò implica che l'ILC può funzionare anche per sistemi di modelli incerti.

1.3.1 Legge di aggiornamento per un sistema PID

Per un sistema PID a tempo continuo la legge di aggiornamento è definita in maniera completamente differente:

$$u_{(k+1)} = u_k + \Phi e_k + \Gamma \dot{e}_k + \Psi \int e_k \, dt \quad (1.13)$$

Dove Φ, Γ e Ψ sono le matrici di guadagno d'apprendimento.

1.4 ILC con perdita di dati randomici

Dopo aver visto i due principali sistemi per progettare un ILC, possiamo descrivere e illustrare i 3 modelli più importanti di perdita di dati elencando le varie caratteristiche e differenze.
La perdita o meno di dati in sistemi descritti nel paragrafo precedente, possono essere paragonati a dei *switch* che aprono e chiudono il canale in maniera randomica, denotata da una variabile randomica $\gamma_k(t)$.
Nello specifico, esistono due tipi di stati: $\gamma_k(t)$ varrà 1 quando il pacchetto di dati è trasmesso correttamente, 0 altrimenti.
I modelli generalmente più usati per la perdita di dati sono:

- Random Sequence Model (RSM)
- Bernoulli Variable Model (BVM)
- Markov Chain Model (MCM)

1.4.1 Random sequence model

$\forall t$, la misura della perdita del pacchetto è randomica senza tener conto di una certa distribuzione di probabilità. Tengo però conto di un $k \geq 1$ tale che in almeno un'iterazione, la misurazione abbia esito positivo e che il pacchetto sia tornato indietro durante le successive k-esime iterazioni.
Il numero k del random sequence model indica che la lunghezza massima delle successive perdite di dati è k-1. Pertanto, il caso k=1 non avrà perdita di dati mentre per k=2 non si verificheranno interruzioni di dati nelle 2 iterazioni successive.
Inoltre, il valore dei numeri di iterazioni k è strettamente correlato al tasso di perdita di dati ma è importante notare che non parliamo di due cose totalmente identiche.
Infatti, il tasso di perdita di dati (denominato *data dropout rate DDR*) può essere formulato come:

$$\lim_{n\to\infty} 1/n \times [\sum_{k=1}^{n}(1-\gamma_k(t))] \qquad (1.14)$$

In altre parole, il DDR denota la media della perdita di dati lungo l'asse d'iterazione mentre k implica il caso peggiore delle successive perdite di dati.
Un valore molto grande di k generalmente corrisponde ad un alto DDR mentre un valore molto piccolo di k corrisponde ad un valore basso di DDR.

Figura 1.2

Nella figura 1.2 possiamo vedere a livello grafico il funzionamento del modello RSM. Sull'asse delle ascisse abbiamo il dominio del tempo t e sulle ordinate il numero delle iterazioni k.
Possiamo vedere che in ogni iterazione sono presenti rettangoli color bianco e dorato che rappresentano rispettivamente i pacchetti persi e i pacchetti correttamente trasmessi. Notiamo come nell'esempio è stato preso k=5 e almeno una misurazione all'istante t risulterà con esito positivo, ovvero almeno un rettangolo dorato sarà presente nel t-esimo istante.

1.4.2 Bernoulli variable model

In questo tipo di modello la variabile randomica $\gamma_k(t)$ è indipendente da ogni istante di tempo t differente e da ogni istante iterativo k. Inoltre, $\gamma_k(t)$ segue la distribuzione di Bernoulli con:

$$\mathrm{P}(\gamma_k(t) = 1) = \bar{\gamma}\ ,\ P(\gamma_k(t) = 0) = 1 - \bar{\gamma} \qquad (1.15)$$
$$\text{Con } 0 < \bar{\gamma} < 1$$

Notiamo come a livello matematico $\bar{\gamma}$ è correlato al *data dropout rate* (DDR) in relazione alla legge dei grandi numeri. Infatti, il DDR è uguale a 1-$\bar{\gamma}$.
Se $\bar{\gamma}$ = 0, implicherà che il sistema è completamento corrotto e inutilizzabile, non ci sono informazioni da poter ricevere dal plant e nessun algoritmo può essere applicato per migliorare le performance di tracciamento.
Se invece $\bar{\gamma}$ = 1, ciò implicherà che nessuna perdita di dati inficerà il sistema e potremmo di conseguenza ricevere informazioni utili dal plant ed è per questo motivo che dobbiamo assumere $0 < \bar{\gamma} < 1$

1.4.3 Markov chain model

La variabile randomica $\gamma_k(t)$ è indipendente per ogni singolo istante di tempo t.
Inoltre, per ogni t, l'evoluzione di $\gamma_k(t)$ lungo l'asse iterativo segue due stati della catena di Markov la cui probabilità della matrice di transizione è scritta come:

$$P = \begin{bmatrix} P_{(11)} & P_{(10)} \\ P_{(01)} & P_{(00)} \end{bmatrix} = \begin{bmatrix} \mu & 1-\mu \\ 1-\delta & \delta \end{bmatrix} \qquad (1.16)$$

Con $0 < \mu, \delta < 1$ e
$$P_{(11)} = (P(\gamma_{k+1}(t) = 1 \mid \gamma_k(t) = 1)$$
$$P_{(10)} = (P(\gamma_{k+1}(t) = 0 \mid \gamma_k(t) = 1)$$
$$P_{(01)} = (P(\gamma_{k+1}(t) = 1 \mid \gamma_k(t) = 0)$$
$$P_{(00)} = (P(\gamma_{k+1}(t) = 0 \mid \gamma_k(t) = 0)$$

Generalmente il modello di Markov viene utilizzato per modellare le perdite di dati a livello generale. Le probabilità di transizione μ e δ denotano il livello medio di mantenimento dello stesso stato per il successo o insuccesso del pacchetto trasmesso.
Se $\mu + \delta = 1$, il modello di Markov verrà convertito nel modello di Bernoulli (Bernoulli variable model). È per questo che il BVM viene considerato come un caso speciale di MCM.

1.5 Differenze tra i tre modelli

Il random sequence model differisce sia da MCM che BVM in quanto non richiede nessuna specifica distribuzione di probabilità o proprietà statistica della variabile randomica $\gamma_k(t)$. Tuttavia, l'RSM porta con sé anche aspetti negativi come il fatto che la lunghezza dei pacchetti persi successivi è limitata rispetto ai due altri modelli.
Nello specifico sia il *bernoulli variable model* che il *markov chain model* ammettono una lunga sequenza di pacchetti persi associata ad un'adeguata probabilità di successo.
Per come è progettato l'RSM, i dati persi non saranno totalmente stocastici; Inoltre, la differenza tra BVM e MCM sta nel fatto che la perdita di dati avviene totalmente lungo l'asse di iterazione per quanto riguarda il modello di Bernoulli mentre ciò non accade nel modello di Markov.
L'indipendenza dalla perdita dei dati ammette alcuni calcoli specifici come la media e la varianza e ne deriva di conseguenza l'analisi della convergenza, questa tecnica non è applicabile per il modello MCM.

1.5.1 Studio sui modelli più utilizzati

Il modello più utilizzato per la perdita di dati è il Bernoulli variable model. Come detto nel paragrafo precedente, la variabile randomica varrà 1 con probabilità di successo pari a $\bar{\gamma}$ e valore uguale a 0 con la probabilità di insuccesso pari a 1-$\bar{\gamma}$.
In altre parole, questo modello ha una chiara e ben definita distribuzione di probabilità e una buona indipendenza dalla perdita di dati.
È per questo che in molte pubblicazioni fanno riferimento al BVM quando si parla di perdita di dati.
Tuttavia, però ci sono degli studi sugli ILC che criticano il modello preso in questione.
Nel paper [1 vedi bibliografia], l'autore da delle informazioni importanti per quanto riguardano gli effetti della perdita di dati. L'autore considera unicamente il caso in cui solo un singolo pacchetto è stato perso durante la trasmissione e successivamente ne

ha descritto la conseguenza degli effetti per quanto riguardano gli errori in ingresso e le performance di tracciamento.
Altri documenti [2-3 vedi bibliografia] utilizzano il *Random sequence model* per la perdita di dati, nello specifico la sequenza di perdita di dati lungo l'asse di iterazione non è stata utilizzata per essere affiancata con nessuna distribuzione di probabilità specifica.
Ciò implica che la proprietà statistica di perdita di dati può variare lungo l'asse di iterazione. Pertanto, la distribuzione stabile nel modello di bernoulli non verrà utilizzata.
Inoltre, per assicurare la convergenza asintotica della sequenza degli input, un nuovo vincolo deve essere imposto; Almeno un valore sufficientemente grande di k tale che durante qualsiasi successiva k iterazione, almeno un'informazione di dropout assumerà valore pari a 1.
In un numero molto ristretto di paper, troviamo l'utilizzo del *Markov chain model.* In questo modello, la perdita di dati dipende principalmente dagli eventi passati mentre la perdita o meno del pacchetto corrente affliggerà la probabilità di successo di trasmissione del pacchetto successivo.

1.5.2 Posizionamento della perdita di dati nell'ILC

Nel sistema ILC, il controllore per l'apprendimento e il plant sono separati e comunicano tra loro a livello fisico attraverso una rete wireless o cablata.
Così facendo ci sono due canali che connettono quest'ultimi; il *primo canale* si trova lato misura (denominato *measurement side*) che trasmette le informazioni di misurazione dell'output al controllore d'apprendimento.
Il *secondo canale* è situato lato attuatore (denominato *actuator side*) per trasmettere il segnale d'ingresso generato dal plant in modo tale che il processo operativo possa essere eseguito in maniera costante e continua.
Quando consideriamo una perdita di informazioni per l'ILC, la posizione in cui la perdita di dati si verifica, è generalmente assunta *lato misurazione.*
In altre parole, solo la rete network *measurement side* è assunta come rete di **tipo lossy**.
La seconda network *actuator side* è pensata per lavorare in maniera ottimale nella maggior parte di sistemi il cui segnale input generato può essere a volte inviato al plant senza alcun tipo di perdita.
Tuttavia, su alcuni papers, si evidenzia il fatto che sia la network lato misurazione che lato attuatore, soffrono di perdita di dati. Nello specifico quando la rete lato misurazione soffre di perdita di dati randomici, il segnale output del plant potrebbe o meno essere trasmesso in maniera corretta.
Un semplice meccanismo che si adotta per trattare le informazioni lato misurazione può essere descritto nel seguente modo: se l'output di misura è trasmesso correttamente, allora il controllore d'apprendimento potrebbe impiegare tali informazioni per l'aggiornamento.
Se l'output di misura è perso durante la trasmissione, allora il controllore potrebbe fermare l'aggiornamento finché le corrispondenti informazioni dell'output non vengono trasmesse in maniera corretta. In questo meccanismo le perdite di informazioni sono

semplicemente rimpiazzate da una lunga stringa di zeri. È per questo motivo che la perdita di dati si verifica solo lato misurazione; D'altronde, quando consideriamo la perdita di dati lato attuatore, è chiaro che la perdita del segnale input non può essere rimpiazzata semplicemente con una stringa di zeri in quanto potrebbe danneggiare notevolmente le performance di tracciamento dell'intero sistema. Infatti, se la network dovesse soffrire di dropout lato attuatore, il segnale perso dovrebbe essere compensato con un pacchetto adatto per mantenere le operazioni di processo del plant.
In alcuni trattati [1 vedi bibliografia] è stato descritto un metodo per compensare queste perdite di dati. Quando un pacchetto del segnale input viene perso verso lato attuatore, l'istante successivo al segnale verrà utilizzato per cercare di compensare tale perdita.
Per riportare un esempio, se l'input all'istante *t* è perso, sarebbe compensato con l'input all'istante di tempo *t – 1*. Anche quando un pacchetto del segnale output viene perso (lato misurazione), un meccanismo di compensazione simile viene applicato.

1.5.3 Significati di convergenza

Arrivati a questo punto possiamo iniziare ad analizzare le analisi tecniche ed i relativi risultati di convergenza a livello matematico delle varie documentazioni studiate.
Ma nello specifico, cos'è la convergenza? Per rispondere a questa domanda partiamo dagli schemi iterativi. Quest'ultimi generano sequenze di input e le *analisi di convergenza,* ci dicono quando e soprattutto, sotto quali condizioni, queste sequenze convergeranno.
Nello specifico, andiamo a capire il significato di convergenza tenendo conto sia del fattore randomico della perdita di dati che dei rumori stocastici.
Una tecnica largamente utilizzata si basa sul *filtro di Kalman* proposta inizialmente dall'azienda **Saab.** Questa tecnica fu applicata e ottenne la convergenza quadratica media della sequenza di input.
Con i recenti progressi sull'ILC in presenza di perdita di dati, possiamo osservare che:

- Nella maggior parte dei documenti, la perdita dei dati è modellata dalla *variabile randomica di Bernoulli* mentre i risultati secondo la sequenza del modello randomico sono piuttosto limitati.
- Quasi tutti i documenti considerano la perdita di dati, verificata proprio nel *lato di misura* mentre una piccolissima parte di essi danno la colpa del dropout al *lato attuatore.*

Model			Position		Convergence			
RSM	BVM	MCM	Measurement	Actuator	M.E.	M.S.	A.S.	D.A.
	•		•			•		
	•		•			•		
	•		•	•		•		
	•		•					•
	•		•	•	•			
	•		•			•		
	•		•			•		
	•		•	•				•
	•		•	•	•			
	•		•		•			
	•		•		•	•	•	
	•		•				•	
	•		•				•	
	•		•	•		•	•	
	•		•	•				•
•			•				•	
•			•				•	

Figura 1.3

Nella figura 1.3 possiamo vedere uno studio effettuato su una diversa mole di papers svolto per rappresentare i risultati di convergenza a livello matematico delle varie documentazioni.
Nelle colonne di convergenza, M.S sta per mean square, ovvero raffigurano tutti quei risultati che hanno avuto una convergenza di tipo quadratica; M.E sta per matematical expectation ovvero i risultati matematici che ci saremmo aspettati, A.S sta per almost sure e D.A sta per analisi deterministica.

1.6 ILC con altre informazioni incomplete

1.6.1 Ritardi di comunicazione e asincronismo

Durante lo studio di progettazione dell'ILC, ho trovato davvero pochi documenti riguardo i problemi di delay e asincronismo.
Il paper [4] considera questo specifico problema per sistemi a tempo discreto dove uno schema della rete *P-type (*definita in maniera rigorosa nel capitolo 1.1.2) è stato preso in esame. Nello schema i dati ritardati sono compensati dai dati provenienti della precedente iterazione e come risultato, i successivi ritardi lungo l'asse di iterazione non saranno ammessi.
[5] Un modello generale di *ritardo di comunicazione* o di *comunicazione asincrona* è costituito da un sistema a larga scala formato a sua volta da molteplici sottosistemi in cui la comunicazione tra differenti sottosistemi soffre in maniera randomica di ritardi di comunicazione e asincronismo.
Nel particolare, per *i* ***sistemi a larga scala***, è difficile per ogni sottosistema assimilare tutte le informazioni dell'intero sistema quando si generano segnali di controllo.
Proprio per questo motivo un *controllo decentralizzato* risulta più pertinente per rimediare al problema che ci viene presentato. A causa dei valori di efficienza potenzialmente diversi tra i vari sottosistemi, l'azione di controllo potrebbe non essere aggiornata per tutti i sottosistemi ad ogni step iterativo. Perciò, aggiornando tutti i sottosistemi potremmo introdurre asincronismo randomico. Notiamo come anche

questo problema è facilmente risolvibile seguendo un metodo molto simile riguardante il problema della perdita di dati descritto nel punto 1.3.1. (modello RSM)
Per quanto riguarda il *ritardo temporale,* questo argomento è stato ampiamente studiato in quanto potrebbe ridurre notevolmente le prestazioni di sistema.
Tuttavia, la vera essenza dell'ILC è proprio quella di "*riparare*" il segnale input utilizzando le informazioni d'ingresso e uscita delle precedenti iterazioni lungo, appunto, l'asse iterativo e non temporale. Così facendo le informazioni ripetitive lungo l'asse iterativo non influenzeranno significatamene l'ILC.
Come abbiamo visto sino ad ora, uno dei principali vantaggi di ILC è quello di ridurre la dipendenza del sistema informativo; In altre parole, un ritardo temporale sconosciuto ma fissato non impatterà sulle performance di controllo finché verrà considerato parte delle informazioni del sistema stesso.

1.6.2 Esempi di utilizzo di modelli ILC

In alcune applicazioni pratiche, l'iterazione d'invarianza intrinseca è spesso violata a causa di incertezze sconosciute o fattori imprevedibili. In questa sottosezione, ci concentriamo sulla necessità di trovare lunghezze 'di prova' identiche tra loro, che sono state ritenute non valide, per esempio, in alcuni sistemi della sfera biomedica.
Mentre applichiamo un controllo ad apprendimento iterativo in una tecnica di stimolazione funzionale per il movimento degli arti superiori e per l'assistenza alla deambulazione, è stato visto che i processi operativi terminano in anticipo almeno per le prime passate (o iterazioni) principalmente per motivi di sicurezza considerando che l'output può deviare in maniera significativa dalla traiettoria desiderata. [6 vedi bibliografia]

1.6.3 ILC con informazioni passive incomplete

In questo paragrafo ci concentriamo sull'approfondimento dell'ILC con informazioni passive incomplete. D'ora in poi per *informazioni passive incomplete,* intendiamo quel gruppo di dati e informazioni incomplete causate da una limitazione del sistema durante la fase di acquisizione, di memorizzazione, di trasmissione e di processo come, per esempio, la perdita di dati trattata sino ad ora, ritardi o limitazione di trasmissione a livello di banda.
Di conseguenza, l'obiettivo principale è proprio quello di analizzare il problema e di creare un corrispondente algoritmo iterativo per colmare tale problema.
Ci si deve soffermare su due aspetti principali il primo dei quali pone il problema su come progettare questo meccanismo di compensazione quando un pacchetto di dati e informazioni vengono persi.
Il secondo punto riguarda il fatto di stabilire una struttura ben definita e unificata per l'analisi di convergenza quadratica media e sicura.

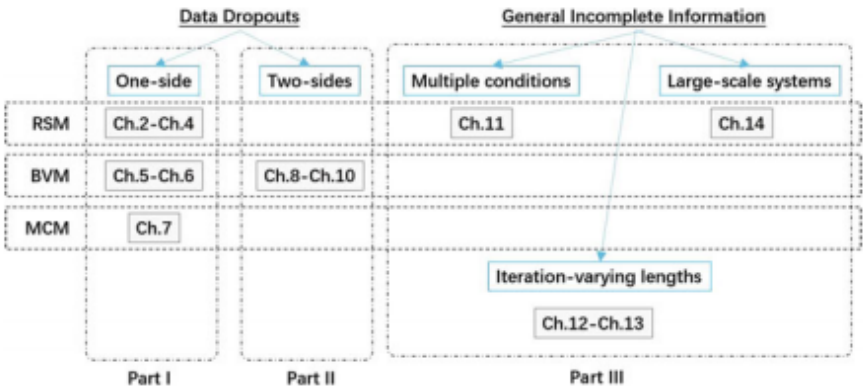

Figura 1.4

Nella figura 1.4, possiamo vedere il grafico di struttura diviso in tre parti; La prima delle tre, punta a fornire informazioni più approfondite sull'ILC sotto diversi tipi di modelli di perdita di dati sempre solo lato misurazione (studiati nei precedenti capitoli).
Nella seconda parte si forniscono informazioni sempre sulla perdita di dati, questa volta sia lato misurazione che attuatore. La terza ed ultima parte mira a dare ulteriori risultati sul carattere dell'ILC in presenza di altre informazioni incomplete studiate proprio in questo capitolo (ritardi e asincronia di comunicazione, lunghezze variabili di iterazione o i problemi su sistemi a larga scala affrontati nel paragrafo 1.5.1).

1.7 Spazi vettoriali

Prendiamo in considerazione i seguenti spazi vettoriali U e Y. Assumiamo U come uno *spazio vettoriale normato* e sia $Y_n \subset Y$ un altro spazio vettoriale normato del sottoinsieme di Y.
Il problema di ILC può essere formulato come segue (vedi figura 1.4); Dato un plant $P: U \to Y$ e un insieme di output desiderati $y_n \in Y_n$, l'obiettivo di ILC è quello di costruire una sequenza di input $u_0, u_1, \ldots$ tale che $y_k := Pu_k$.
Possiamo dire quindi che:

1. $u_k \in U \; \forall k \geq 0$, ed esisterà $\bar{u} \in U$ tale che il $lim_{k\to\infty} u_k = \bar{u}$
2. $y_k \in Y \; \forall k \geq 0$, ed esisterà $\bar{y} \in Y_n$ tale che il $lim_{k\to\infty} y_k = \bar{y}$

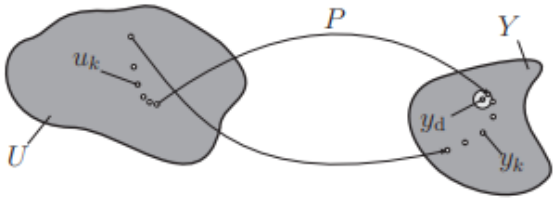

Figura 1.5

Nello specifico, nella figura 1.5 dato il plant $P: U \rightarrow Y$ e $y_d \in Y$, il problema di ILC è quello di costruire input u_k tale che possa corrispondere ad una sequenza di output y_k in modo tale da convergere nelle immediate vicinanze di y_d.
In altre parole, il problema di ILC è quello di costruire un'iterazione per cui esisterà al più un $\bar{u} \in U$ e un $\bar{y} \in Y_n$ tale che $lim_{k\rightarrow\infty} Pu_k = \bar{y}$.
Così facendo, l'errore $\left|\left|y_d - \bar{y}\right|\right|_{Y_n}$ sarà estremamente piccolo.

1.8 Filtro di kalman

Nel paragrafo 1.4.3 abbiamo parlato di questo filtro per arrivare ai risultati di convergenza per sistemi con perdita di dati. *Il filtro di kalman* è uno speciale filtro che consente di stimare lo stato di un sistema a partire dai dati misurati; In altre parole, è un algoritmo ricorsivo per l'elaborazione di dati.
L'algoritmo del filtro consiste in un processo suddiviso in due parti distinte: la prima parte esegue una previsione sullo stato dell'intero sistema, mentre la seconda utilizza misurazioni del rumore per perfezionare la stima sullo stato del sistema; I filtri di Kalman potranno essere utilizzati sia in sistemi MIMO che SISO.
Questo filtro può essere visto come un blocco che acquisisce segnali affetti da rumore in ingresso per restituire segnali in uscita affetti da rumore ridotto.
È necessario però fornire una descrizione statistica delle componenti randomiche del sistema.
Tipicamente queste componenti vengono descritte con il modello del rumore bianco oppure gaussiano a media nulla.
Si preferiscono utilizzare questi due modelli perché sia descrivono bene il rumore presente in molti processi fisici, sia perché costituiscono un modello efficace e semplice da implementare.
L'idea fondamentale alla base del filtro è quella di migliorare le stime fornite dal plant, con misure che vengono fornite al passaggio di ogni iterazione.
Il contributo predittivo è basato sulla stima dell'errore dell'iterazione precedente $k - 1$.
Il contributo correttivo, invece, si basa sul calcolo dei *residui,* dove ogni residuo è definito come la differenza tra la grandezza fornita dalla generica misura e la stessa grandezza stimata dal sistema tramite il modello.

Capitolo 2: Apprendimento e adattamento di un ILC

In questo capitolo ci soffermiamo in maniera più dettagliata sul problema generale del controllo d'apprendimento.
Infatti, prima di iniziare a scendere nel dettaglio e descrivere tutti i modelli randomici che si possono utilizzare in ILC, voglio soffermarmi sul mero fatto della parola *apprendimento,* cos'è e come funziona e se è davvero essenziale nei sistemi ILC.
Se considerando gli algoritmi discussi in precedenza, è lecito chiederci: in quale modo questi modelli studiati implementano un comportamento di "apprendimento"?
Il problema chiave è la loro presunta capacità di adattamento; Il punto focale è l'uso di ogni singola iterazione in relazione all'incertezza del *plant (*ovvero quante informazioni preliminari abbiamo riguardo l'impianto).
Siamo d'accordo che anche se questi algoritmi migliorano con successo le prestazioni ad ogni istante iterativo, non necessariamente "imparano". In altre parole, questo miglioramento prestazionale potrebbe esser dipeso dalla conoscenza dettagliata del plant.
Identifichiamo ciò come una caratteristica tipica di una legge di aggiornamento ben strutturata e proprio per questo siamo portati a suddividere le leggi di aggiornamento in differenti livelli. Questo vuol dire considerare una legge di aggiornamento che aggiorna a sua volta una legge di aggiornamento di livello inferiore per avere un risultato ancora più preciso. In questo capitolo vedremo un paio di leggi di aggiornamento di *livello 1* mostrando le loro relative abilità di adattamento.
Quando ci riferiamo al concetto di iterazione, ci soffermiamo il più delle volte ad un concetto di perfezionamento delle prestazioni lungo l'asse iterativo.
In questo capitolo cercheremo di vedere un punto di vista differente, non necessariamente in conflitto col modello classico, in cui l'uso dell'iterazione k è spiegata in termini di bisogno di conoscenza dell'informazione, derivante da una mancata acquisizione informativa a priori.

2.1 Il problema del controllo d'apprendimento

2.1.1 Prestazioni, robustezza delle prestazioni e apprendimento

Mentre affrontiamo la questione dell'apprendimento, dobbiamo considerare almeno due aspetti.
Per prima cosa dobbiamo guardare le prestazioni a 360 gradi; È giusto dire che, rispetto ai mezzi convenzionali, questi algoritmi migliorano le prestazioni? La risposta è senza dubbio sì.

Ma per quando riguarda la robustezza di quest'ultime, possiamo affermare la stessa cosa? In questo caso la risposta è *no* e ne vediamo subito il motivo.
Se noi assumiamo per certo che conosciamo in maniera approfondita il plant ***P***, questa ipotesi è evidentemente forte e forse anche restrittiva; Dobbiamo vedere se allentando la conoscenza completa del plant, comprometteremo anche le prestazioni del sistema.
Se volessimo allentare a priori le conoscenze dell'impianto, quante conoscenze pregresse dovremmo avere per far convergere l'algoritmo? E soprattutto possiamo rendere il sistema più robusto e quindi farlo soffrire di meno a perdite ipotetiche di dati senza compromettere le performance? Iniziamo a dare delle risposte a queste domande.

2.1.2 Perché iteriamo in caso della conoscenza completa del plant?

Andando più nel dettaglio, il problema della piena conoscenza del plant è di affare numerico piuttosto che di iterazione vera e propria.
L'interazione col plant è una delle caratteristiche che definiscono il controllo d'apprendimento; I dati numerici servono a indirizzare l'intero impianto verso un comportamento desiderato e prestabilito.
Nel caso della sua piena conoscenza, non c'è niente che possiamo imparare rispetto a ciò che già sappiamo e, di conseguenza, non necessiteremo di alcun apprendimento di controllo.

2.1.3 Robustezza delle prestazioni

Prendiamo il caso in cui allentiamo le conoscenze iniziali del plant, cosa succederebbe? Se c'è una mancata corrispondenza tra l'attuale impianto e il modello teorizzato, l'algoritmo continuerà a convergere?
È difficile dare una risposta concreta a quanto detto, ma cerchiamo di rispondere attraverso un esempio.
Supponiamo che l'attuale plant P sia dato da $P \coloneqq P_0(1+\delta_M)$ in cui P_0 è l'impianto teorico, δ_M è una perturbazione limitata.
Affinché la legge di aggiornamento converga, deve essere soddisfatta la seguente condizione (già vista in maniera similare nel punto 1.8):

$$\left\| 1-\frac{(P_0^* P_0)}{(\gamma^2+P_0^* P_0)}(1+\delta_M) \right\| < 1 \qquad (2.1)$$

Ciò implica che il termine $(1+\delta_M)$ dovrà essere reale positivo. Possiamo anche osservare che non può essere soddisfatto nel caso in cui P abbia due o più poli.

Se questo renderà l'algoritmo sufficientemente robusto o meno dipenderà principalmente dall'applicazione specifica.
Il fattore fondamentale per noi da analizzare è riferito ai plant che soddisferanno la condizione 2.1. In questo caso l'errore convergerà a zero mentre per quelli che non lo soddisfano, l'errore divergerà a infinito.
Per costruzione, la convergenza implica la prestazione; A prima vista, ciò sembra essere in contrasto col la situazione nel controllo del feedback dove le prestazioni e la robustezza sono tipicamente scambiati.
Tuttavia, questa discrepanza sparisce dal momento in cui si considerano leggi di aggiornamento generalizzate come quella nel punto 2.2, essendo questa una versione più robusta della legge di aggiornamento classica.

$$u_{k+1} = Q(u_k + (\gamma^2 I + P^* P)^{-1} P^* e_k) \qquad (2.2)$$

Dal punto 2.2 possiamo notare se il design progettato per l'impianto è ottimale; In tal caso ci aspetteremo ottimi risultati a livello di performance ogni qual volta che l'impianto effettivo è il più similare all'impianto teorizzato.
Al contrario, non ci aspetteremo nessun tipo di performance quando i due impianti si discostano completamente e l'algoritmo non convergerà più.
La robustezza della stabilità può essere aumentata abbassando il guadagno di Q; Tenendo in considerazione l'algoritmo citato nel *capitolo 1,* possiamo affermare che per avere una buona performance il guadagno di Q deve essere unitario e di conseguenza valere circa uno.
In altre parole, non possiamo pretendere una buona performance di sistema e un'ottima robustezza del sistema allo stesso tempo, si deve cercare un compromesso tra le due situazioni.

2.1.4 Controllo d'apprendimento

Non possiamo che concludere che, come nel controllo convenzionale del feedback, la troppa incertezza rappresenta una minaccia, poiché abbiamo visto come non sia possibile ottenere prestazioni ottimali senza compromettere la stabilità del sistema.
Il nostro algoritmo risulta essere altrettanto sensibile alla mancanza di conoscenza a priori come lo è il controllo convenzionale.
Questo è in netto contrasto con il fatto che, oltre alle conoscenze pregresse, l'algoritmo descritto nel primo capitolo, dovrà gestire una grande mole di dati disponibili fin dall'inizializzazione dell'informazione stessa.
Quindi se non è la sua abilità nel gestire l'incertezza, allora cos'è che rende questo algoritmo essenziale per il controllo d'apprendimento?

2.2 Adattamento, iterazione e apprendimento

Siamo arrivati al punto di capire che il problema cardine del controllo d'apprendimento nel modo in cui lo abbiamo visto, si basa più sull'incertezza dell'impianto che sulle sue effettive prestazioni.

2.2.1 Incertezza: il problema del controllo d'apprendimento

In ILC cerchiamo di migliorare le prestazioni del sistema attraverso una seri di prove (automatizzate) che si evolvono lungo l'asse iterativo.
Questo sembra funzionare in maniera più che ottimale fino a quando supponiamo di conoscere a priori il *plant* in maniera completa. Ma quando questa affermazione viene a mancare, ovvero la conoscenza a priori dell'impianto non è completa cosa succede?
Col passare di iterazione in iterazione, l'algoritmo compenserà la mancanza di questa conoscenza a priori fino ad arrivare ad un risultato più o meno ottimale; Però, questa affermazione abbiamo visto non essere così ovvia seguendo quanto detto nei precedenti paragrafi.
Come aggiorniamo l'input corrente, se non possiamo (oppure solo approssimativamente) prevedere gli effetti di cambiamento da parte dell'input sull'output del sistema?
Assumendo un'estrema incertezza da parte del plant, il seguente esempio potrebbe aiutarci a comprendere al meglio tale problema.
Prendiamo un determinato plant *P* di cui conosciamo poche informazioni; Poiché il nostro obiettivo è sempre quello di conoscere per intero l'impianto, decidiamo attraverso un opportuno algoritmo di effettuare una serie di prove in modo tale da racimolare più informazioni per far convergere l'intero sistema.
Per ogni prova applichiamo un determinato ingresso u_k, facendo restituire dal sistema l'output corrispondente $y_k = Pu_k$.
Attraverso le varie leggi di aggiornamento *P-type o D-type (paragrafo 1.2.3),* i dati vengono memorizzati in modo tale da usarli per poter migliorare il sistema nell'iterazione successiva attraverso le conoscenze pregresse acquisite nell'istante iterativo precedente.
Diciamo che nel corso delle varie prove, riusciamo a raccogliere una grande quantità di informazioni *D* per cui:

$$D := \{(u_i, y_i), i = 1,2, \dots\} \qquad (2.3)$$

Ma questi dati raccolti, cosa ci dicono esattamente sul nostro sistema?

Questo, ovviamente, dipende dal sistema preso in esame. Per un sistema LTI (sistema lineare tempo invariante), ci basta circa una iterazione per caratterizzare il comportamento dell'input e output del sistema completo.
In questo caso il problema del controllo d'apprendimento degenererà fino a quando non ci sarà bisogno di ripetute iterazioni.
Parlando per sistemi generali, l'insieme dei dati raccolti ad ogni passata iterativa, costituisce un sottoinsieme del carattere del sistema. Inoltre, questo sottoinsieme non è necessariamente rappresentativo, del comportamento totale del sistema.
Perciò, quando si tratta di prevedere la sua risposta per un input che non fa parte di questo sottoinsieme di dati, oppure, ci viene richiesto di ricostruire un input (non facente parte del sottoinsieme di dati) a partire da un corrispondente output, i dati acquisiti potrebbero non essere troppo utili per i fini del sistema stesso.
Quello che manca quindi, è proprio l'inserimento di un corretto modello; Un modello che possa essere adatto ai dati acquisiti nei vari passaggi iterativi e che ci dia una garanzia assoluta sul comportamento totale del sistema.

2.2.2 Iterazioni e apprendimento

Per rendere quanto detto nei precedenti paragrafi il più fruibile possibile, diamo un'altra occhiata alle leggi di aggiornamento che abbiamo descritto nel precedente capitolo, nello specifico prendiamo la formula riportata nel punto 1.4 e la riscriviamo nel seguente modo:

$$u_{k+1} = Qu_k + Le_k \qquad (2.4)$$

Ma cosa rappresenta nel dettaglio questa equazione? Tecnicamente, rappresenta la legge che governa l'evoluzione del sistema iterativo.
È una maniera alternativa per dire come generare in maniera automatizzata, un nuovo input dai dati raccolti dalle precedenti iterazioni k-esime.
Supponiamo che questo algoritmo mostri un comportamento intelligente; Dove risiederebbe? La risposta è facile darla, e risiede nello specifico nei parametri aggiunti $\boldsymbol{Q}$ ed $\boldsymbol{L}$, in questo caso sdoppiati per vedere i corrispondenti comportamenti per quanto riguardano gli input e gli errori nelle iterazioni precedenti.
Se però prendiamo in esempio un sistema causale e diamo per assodato che l'algoritmo preso in esame convergerà, allora qualunque cosa sia stata appresa, dovrà esser mostrata nei punti prefissati dell'equazione:

$$(I - \boldsymbol{Q} + \boldsymbol{L}P)\bar{u} = y_d \qquad (2.5)$$

La quantità $\bar{u}$ è un qualcosa che non conosciamo e che generalmente non possiamo risolvere (data una conoscenza di base del plant P) e quindi se qualcosa è stato appreso, dovrà risiedere nello specifico nella variabile $\bar{u}$.
Notiamo però come il valore sopracitato, è determinato unicamente da Q, L e P e non dall'evoluzione dell'input u_k.
Infatti, così facendo, non ci sorprendiamo se in questo contesto causale, possiamo recuperare la stessa quantità senza l'apporto di una singola iterazione, ovvero senza utilizzare un controllo d'apprendimento.
L'apprendimento e l'iterazione non sono la stessa cosa e non vanno di pari passo. In altre parole, l'iterazione non implica l'apprendimento e viceversa.

2.2.3 Adattamento

Abbiamo visto come la distinzione tra iterazione e apprendimento dipende principalmente da come i dati vengono utilizzati, se attraverso l'evoluzione degli stati precedenti dell'input u_k o senza di essi.
Gli algoritmi sopra citati come per esempio nei punti 2.4 o 2.5, sono evidentemente basati sulla mole di dati acquisiti.
I dati, infatti, vengono utilizzati ma non necessariamente in maniera strumentale; Nella nostra legge manca proprio un elemento che ci dia la possibilità di interpretare e adattare i dati prodotti. Se i dati 'invalidano' il modello su cui si basano, la legge di aggiornamento continuerà ad aggiornare l'input ignara di ciò che sta realmente accadendo. In altre parole, il nostro algoritmo *non* si adatterà.
Supponiamo allora di consentire ai parametri **Q** ed **L** (punto 2.5) di dipendere esclusivamente dai dati. Così facendo, considereremmo il nostro algoritmo come adattivo? In altre parole, l'algoritmo riuscirà a adattarsi in caso di variazioni e/o con la presenza di perdite di dati? Dipende, nello specifico la dipendenza è data dal fatto di possedere una regola generale per rendere esplicita tale dipendenza.
Una volta definito ciò, torneremo al problema iniziale ma con la differenza che ora ci troviamo ad un livello superiore della legge di aggiornamento che aggiornerà la regola di aggiornamento del livello precedente. (Vedi paragrafo 2 in cui abbiamo ampiamente discusso sui vari livelli di legge di aggiornamento).
La ricorsività è evidente e di conseguenza, l'esistenza di un algoritmo adattivo diventa incerta.
D'altronde se ci limitiamo ad un livello per la quale l'input di controllo è sempre aggiornato, allora si, potremmo considerare tale algoritmo anche *adattivo.*

Capitolo 3: Modelli randomici per sistemi lineari

In questa sezione ci concentriamo sul caso in cui la perdita di dati si verifichi solo sul lato output. In altre parole, assumiamo che il canale di comunicazione lato attuatore funzioni bene senza nessun tipo di problema.
In tal caso, il problema principale al quale dovremmo prestare sufficiente attenzione è utilizzare tecniche di analisi adeguate per i vari modelli di dropout.
Nel particolare utilizzeremo i modelli studiati nel capitolo uno per sistemi lineari e non lineari.

3.1 Formulazione del problema SISO

Consideriamo il sistema *Siso (**single input – single output)*** a tempo variante di un **sistema lineare:**

$$x_k(t+1) = A_t x_k(t) + \boldsymbol{B}_t u_k(t) + w_k(t+1) \qquad (3.1)$$
$$y_k(t) = \boldsymbol{C}_t x_k(t) + v_k(t)$$

Formulazione base del sistema di controllo

Dove $t \in 0,1,2,..,N$ denota l'istante di tempo di una precisa iterazione del processo. Con k identifichiamo le differenti iterazioni; $u_k(t) \in R$, $x_k(t) \in R^n$ e $y_k(t) \in R$ identificano rispettivamente l'input, lo stato del sistema e l'output mentre A_t, B_t, C_t denotano le informazioni sconosciute del sistema e $w_k(t)$ e $v_k(t)$ corrispondono rispettivamente al *rumore del sistema stocastico* ed al *rumore di misurazione*.
Per comodità studiamo casi con perdite di dati solo lato misurazione, i quali, possono essere estesi al caso in cui sia i segnali di misurazione che quelli di controllo, abbiano una perdita di dati; Tali tecniche le affronteremo più in là nel capitolo 3.
Come riportato nella figura 3.1, i segnali di misurazione sono ritrasmessi attraverso un *canale lossy,* il quale può essere assimilato ad uno switch che apre e chiude il canale in maniera randomica.

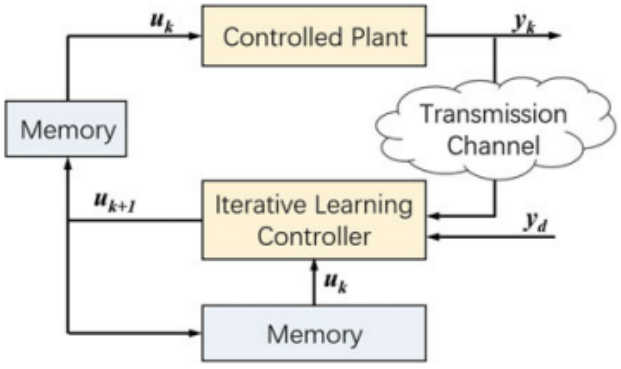

Figura 3.1: schema a blocchi di un sistema di controllo con misurazione dei pacchetti persi

Scendendo più nel dettaglio, in questo capitolo affronteremo la perdita di dati modellandola col *random sequence model*. Denotiamo con M_k l'insieme randomico delle posizioni temporali in cui le misurazioni vengono perse nella k-esima iterazione. In altre parole, $t_0 \in M_k$ se l'output $y_k(t_0)$ è perso.
L'obiettivo di controllo di un ILC per un sistema privo di rumore è quello di costruire una *legge di aggiornamento* che possa generare una sequenza input che garantisca il tracciamento asintoticamente preciso alla traiettoria di riferimento desiderata; Quindi, l'output $y_k(t)$ ha il compito di tracciare la traiettoria $y_d(t)$ in maniera asintotica per gli istanti di tempo specificati.
Quando ci rapportiamo con sistemi stocastici, è impossibile raggiungere questo obiettivo di controllo per via della presenza di questi rumori $w_k(t)$ e $v_k(t)$ definiti poche righe fa.
Non possiamo aspettarci che $y_k(t) \rightarrow y_d(t)$, $\forall t$, per sistemi stocastici per cui l'iterazione $k \rightarrow \infty$.
Notiamo come il rumore stocastico non può essere eliminato da nessun algoritmo per cui il miglior risultato che possiamo aspettarci dal sistema di controllo è quello di assicurare che la *traiettoria di riferimento* desiderata possa essere tracciata con precisione dall'output riducendo il più possibile questo tipo di rumore
In questo capitolo, l'obiettivo del controllo è quello di trovare una sequenza opportuna $\{u_k(t), k = 1,2, \dots, \subset U\}$ con perdita randomica di dati tale che si possano ridurre al minimo i seguenti errori di tracciamento medi avendo così un *controllo ottimale*:

$$V(t) = \lim_{n\to\infty} s\,up\ 1/n \sum_{k=1}^{n} \left||y_k(t) - y_d(t)|\right|^2 \quad (3.2)$$

Ricordo invece che, con $y_d(t)$, $t \in \{0, 1, \dots, N\}$ (3.3), definiamo sempre *l'obiettivo di controllo (*inerente alla traiettoria di riferimento).

3.1.2 Schema di aggiornamento per sistemi SISO

Nell'ultimo paragrafo abbiamo solo definito il *controllo ottimale* poiché le informazioni del sistema erano sconosciute. Pertanto, la *P-type update law* descritta e definita nel capitolo precedente, viene usata con un calo del guadagno d'apprendimento:

$$u_{k+1}(t) = u_k(t) + a_k 1_{[(t+1)\notin \boldsymbol{M_k}]}\big(y_d(t+1) - y_k(t+1)\big) \quad (3.4)$$

Definisco a_k il *guadagno decrescente* con $a_k > 0$; 1_{evento} è una funzione che varrà 1 se l'evento indicato tra parentesi è soddisfatto, zero altrimenti.
Per iniziare ad analizzare la convergenza della legge d'aggiornamento (3.4), dobbiamo prima riscrivere come segue il sistema in forma vettoriale con una tecnica chiamata 'tecnica di rialzo ' (lifting technique).

$$U_k = [u_k(0), u_k(1), \ldots, u_k(N-1)]^T \in R^N \qquad (3.5)$$
$$Y_k = [y_k(1), y_k(2), \ldots, y_k(N)]^T \in R^N$$

Inoltre, Y_k viene utilizzata a relazionare l'input con l'output nella seguente maniera:

$$Y_k = HU_k + y_d^0 + w_k \quad (3.6)$$

Nello specifico, $y(t), U(t) e\ w(t)$ son una realizzazione di un processo aleatorio che rappresentano rispettivamente il segnale ricevuto, il segnale inviato ed il rumore stocastico.
Con y_d^0, cediamo le responsabilità delle condizioni iniziali del sistema e lo si definisce nel seguente modo:

$$y_d^0 = \left[\left(c_1 A_0 x_d(0)\right)^T, \left(c_2 A_1 A_0 x_d(0)\right)^T, \ldots, \left(c_N \sum_{L=0}^{N-1} A_K x_d(0)\right)^T\right]^T \qquad (3.7)$$

Il termine w_k invece è considerato come un processo rumoroso a termine nullo gaussiano formato da una matrice di covarianza *Q*. Per riportare un esempio $w_k \sim N(0, Q)$ (3.8).
D'altronde possiamo definire y_d come la somma di $y_d = HU_d + y_d^0$ dove $y_d\ e\ U_d$ sono definiti in maniera similare a $Y_k\ e\ U_k$.
Per finire, la *legge di aggiornamento* può essere riscritta come:

$$U_{k+1} = U_k + a_k \Gamma_k (\mathrm{y_d} - \mathrm{y_k}) \quad (3.9)$$

Γ_k è rappresentato con la seguente matrice:

$$\Gamma_k = \begin{bmatrix} 1_{[1 \notin M_k]} & 0 & \cdots & 0 \\ 0 & 1_{[2 \notin M_k]} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1_{[N \notin M_k]} \end{bmatrix} \qquad (3.10)$$

E H essendo una matrice delle risposte impulsive viene definito nella seguente maniera:

$$H = \begin{bmatrix} c_1 b_o & 0 & \cdots & 0 \\ 2_2 A_1 b_0 & c_2 b_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ c_N A_{N-1} b_0 & \cdots & \cdots & c_N b_{N-1} \end{bmatrix} \qquad (3.11)$$

3.2 Formulazione del problema MIMO

In questo paragrafo estendiamo le nozioni assimilate del sistema *Siso* del paragrafo 3.1 per sistemi '*multi-input multi-output'(Mimo*)**.** Inoltre, consideriamo anche il grado arbitrario del sistema; Nel particolare utilizzeremo il seguente sistema lineare tempo variante:

$$x_k(t+1) = A_t x_k(t) + B_t u_k(t) + w_k(t+1) \quad (3.12)$$
$$y_k(t) = C_t x_k(t) + v_k(t)$$

Il pedice k viene utilizzato per indicare il numero di iterazioni, $k = 1, 2, \ldots, t, \forall t \in 0,1,2,\ldots,N,$ dove N identifica la lunghezza di iterazione.
Le variabili $x_k(t) \in R^n$, $u_k(t) \in R^p$ e $y_k(t) \in R^q$, rappresentano rispettivamente lo stato del sistema, l'input e l'output.
Le notazioni $w_k(t) \in R^n$ e $v_k(t) \in R^q$, identificano invece il sistema e i rumori di misurazione mentre $A_t,\ B_t\ e\ C_t$ sono sistemi matriciali di appropriate dimensioni.
Con questo tipo di sistema assumiamo che il grado relativo sia τ tale che:

$$\tau \geq 1, \quad \forall t \geq \tau$$

Per fare più chiarezza, il grado relativo implica il ritardo minimo di struttura dell'input che inciderà successivamente sul corrispondente output.
Se per esempio prendiamo un $\tau = 1$, l'input nell'istante temporale *t,* avrà un effetto sull'output nell'istante di tempo *t + 1* e nessuna ripercussione nell'istante *t.*
Inoltre, assumendo che il grado relativo sia pari a τ, facciamo partire le operazioni nell'istante $t = \tau$ in modo tale da guidare l'input $u_k(0)$.
In altre parole, stiamo dicendo che l'output dal tempo t = 0 fino a $t = \tau - 1$ sarà incontrollabile agendo in questa maniera.
Come conseguenza, questi output dovranno esser formulati nelle condizioni iniziali; In aggiunta, considerando la formulazione data per i sistemi *Mimo,* il grado relativo potrebbe variare a seconda della grandezza del vettore output, cioè, a diverse dimensioni dell'output saranno assimilati diversi valori di grado relativo.

3.2.1 Traiettoria di riferimento per sistemi MIMO

Iniziamo ora a identificare la traiettoria desiderata come $y_d(t), t \in 0,1,\ldots,N$.Senza considerare perdite di qualsiasi genere, assumiamo che la traiettoria di riferimento sia realizzabile; Ovvero, con un appropriato valore iniziale di $x_d(0)$, esisterà un valore unico di input $u_d(t)$ per cui:

$$x_d(t+1) = A_t x_d(t) + B u_d(t) \quad (3.13)$$
$$y_d(t) = C_t x_d(t)$$

Come per il sistema *Siso,* $w_k(t)$ *e* $v_k(t)$ sono solitamente indipendenti lungo l'asse di iterazione. u_d e y_d possono essere definite rimpiazzando semplicemente il pedice *k* col pedice *d* poiché tra loro, ci sono ben poche differenze.
In aggiunta, il sistema per il quale è applicabile il metodo ILC dovrebbe esser ripetuto in modo tale che le prestazioni di tracciamento possano esser gradualmente migliorate lungo l'asse di iterazione avvalorando così, l'affermazione fatta poco fa per cui tutti i rumori stocastici sono generalmente indipendenti dall'asse iterativo.

3.2.2 Schema di aggiornamento per sistemi MIMO

I rumori stocastici $w_t(t)$ e $v_k(t)$ sono a tutti gli effetti delle sequenze facenti parte della cosiddetta *sequenza di Martingala,* si evolvono sempre sull'asse di iterazione con condizioni ben definite fin dall'inizio.
Ciò ne deriva che i rumori stocastici sono generalmente indipendenti lungo l'asse di iterazione e che il classico rumore bianco soddisfa questa ipotesi.
Come per la formulazione del problema *MIMO,* diamo una forma a quanto detto per scrivere in forma vettoriale la tecnica utilizzata denominata '*tecnica di rialzo'* del sistema preso in esame nel punto 3.12.

$$U_k = [u_k^T(0), u_k^T(1), \dots, u_k^T(N-\tau)]^T \quad (3.14)$$
$$Y_k = [y_k^T(\tau), y_k^T(\tau+1), \dots, y_k^T(N)]^T \quad (3.15)$$

In maniera molto simile a come già visto per il sistema MIMO, U_d e Y_d possono essere definite rimpiazzando semplicemente il pedice *k* delle equazioni 3.14 e 3.15, con il pedice *d.*
La matrice di trasferimento *H* può essere invece formulata come segue:

$$H = \begin{bmatrix} C_t A_1^{\tau-1} B_0 & 0_{qxp} & \cdots & 0_{qxp} \\ C_{\tau+1} A_1^{\tau} B_0 & C_{\tau+1} A_2^{\tau} B_1 & \cdots & 0_{qxp} \\ \vdots & \vdots & \ddots & \vdots \\ C_N A_1^{N-1} B_0 & C_N A_2^{N-1} B_1 & \cdots & C_N A_{N-\tau+1}^{N-1} B_{N-\tau} \end{bmatrix} \quad (3.16)$$

Per quanto riguarda la relazione tra l'input e l'output, verrà definita nel seguente modo:

$$Y_k = HU_k + Mx_k(0) + \xi_k \quad (3.17)$$

$$Y_d = HU_d + Mx_d(0)$$

Dove $M\ e\ \xi_k\ sono\ rispettivamente$:

$M = [(C_\tau A_0^{\tau-1})^T, \dots, (C_N A_0^{N-1})^T]^T$ e

$$\xi_k = \left[\left(\sum_{i=1}^{\tau} C_\tau\, A_i^{\tau-1} w_k(i) + v_k(\tau)\right)^T, \dots, \left(\sum_{i=1}^{N} C_N\, A_i^{N-1} w_k(i) + v_k(N)\right)^T\right]^T \qquad (3.18)$$

Inoltre, l'errore di tracciamento sarà sempre dato da $e_k(t) = y_d(t) - y_k(t)$; Denotiamo anche *l'errore per la tecnica di rialzo* nella seguente maniera: $E_k \triangleq Y_d - Y_k$.
È evidente che:

$$E_k \triangleq Y_d - Y_k = \boldsymbol{H(U_d - U_k)} - \xi_k \qquad (3.19)$$

3.2.3 Sequenza di Martingala

Nel paragrafo 3.2.2 abbiamo accennato della *sequenza di Martingala* per i rumori stocastici del sistema MIMO.
Nella teoria delle probabilità la sequenza di martingala è un processo stocastico caratterizzato da un valore atteso nullo.
In altre parole, un processo stocastico generale è definito come:

$$\varepsilon = \{\varepsilon_t\}_{t\in T} \qquad (3.20)$$

Tale processo può essere identificato come sequenza di Martingala se vale la seguente condizione:

$$E(\varepsilon_t|\varepsilon_{t-1}, \varepsilon_{t-2}, \dots) = 0\ \forall t \in T \qquad (3.21)$$

Nel caso in cui il valore atteso non dipenda esclusivamente dall'iterazione precedente della variabile, ma da un altro insieme di informazioni X, si dirà che il processo è una *differenza di martingala condizionatamente ad X.*

3.3 Caso di un sistema senza rumore

In questa sezione, consideriamo il caso in cui il rumore stocastico è completamente assente; In altre parole, prendiamo in esame il sistema proposto nel punto 3.13 e lo consideriamo in assenza di rumore.

$$x_k(t+1) = A_t x_k(t) + B u_k(t) \qquad (3.22)$$
$$y_k(t) = C_t x_k(t)$$

Per questo tipo di sistema, il fattore randomico è dato solo dalla perdita di dati effettiva, il quale ci fornisce una visione concisa per affrontare le influenze della perdita di dati e dei rumori stocastici.
La legge di aggiornamento di tipo *P-type* viene ridefinita come segue:

$$u_{k+1}(t) = u_k(t) + \rho\gamma_k(t+\tau)L_t e_k(t+\tau) \qquad (3.23)$$

Per $t = 0, \dots, N-\tau$, la variabile ρ è una costante positiva che analizzeremo successivamente, per quanto riguarda $L_t \in R^{pxq}$, quest'ultima è una matrice di guadagno d'apprendimento utilizzata principalmente per regolare la direzione del controllo.
Inoltre, $\gamma_k(t+\tau) \triangleq 1_{t+\tau \notin M_k}$; Per $\gamma_k = 1$, il corrispondente pacchetto sarà trasmesso correttamente mentre per $\gamma_k = 0$ il pacchetto sarà andato perso.
Per inciso, tengo a ricordare di nuovo che per i sistemi ILC, la legge di aggiornamento non è vincolata unicamente alla *P-type.* D'altronde ci soffermiamo su questa legge perché è più fruibile e concisa.
È anche evidente che per come è stata definita la variabile ρ, quest'ultima può essere semplicemente integrata nella matrice di guadagno L_t.
Tuttavia, in questo paragrafo forniamo la corretta formulazione di ogni singola variabile in maniera separata facendo anche una comparativa nel paragrafo successivo con il caso del sistema stocastico in presenza di rumore.
Come abbiamo fatto nelle formulazioni 3.14 e 3.15, "alziamo" l'input lungo l'asse temporale. La legge di aggiornamento definita nel punto 3.23 può essere riscritta nella seguente maniera:

$$U_{k+1} = U_k + \rho\Gamma_k L E_k \qquad (3.24)$$

Dove $E_k = Y_d - Y_k$,definito precedentemente nel punto 3.19 con un $\xi_k = 0$, è il vettore degli errori di tracciamento per $t = \tau, \dots, N$ mentre Γ_k e L sono definiti come segue:

$$\Gamma_k = \begin{bmatrix} \gamma_k(\tau) I_q & & & \\ & \gamma_k(\tau+1) I_q & & \\ & & \dots & \\ & & & \gamma_k(N) I_q \end{bmatrix} \qquad (3.25)$$

$$L = \begin{bmatrix} L_0 & & & \\ \vdots & L_1 & & \\ & & \dots & \\ & & & L_{N-\tau} \end{bmatrix} \qquad (3.26)$$

3.3.1 Caratteristiche delle matrici

È utile specificare che $\Gamma_k = diag\{(\gamma_k(\tau), (\gamma_k(\tau+1), \dots, (\gamma_k(N)\} \otimes I_q$.
Ricordando come avevamo definito $E_k = H(U_d - U_k)$ e sostituendola nell'equazione del punto 3.24, alla fine otterremo:

$$U_{k+1} = U_k + \rho \Gamma_k LH(U_d - U_k) \qquad (3.27)$$

Ora possiamo definire $\Lambda_k \triangleq \Gamma_k LH$ e nello specifico:

$$LH = \begin{bmatrix} L_0 C_\tau A_1^{\tau-1} B_0 & 0_p & \cdots & 0_p \\ L_1 C_{\tau+1} A_1^{\tau} B_0 & L_1 C_{\tau+1} A_2^{\tau} B_1 & \cdots & 0_p \\ \vdots & \vdots & \ddots & \vdots \\ L_{N-\tau} C_N A_1^{N-1} B_0 & L_{N-\tau} C_N A_2^{N-1} B_1 & \cdots & L_{N-\tau} C_N A_{N-\tau+1}^{N-1} B_{N-\tau} \end{bmatrix} \qquad (3.28)$$

Poiché **LH** è una matrice triangolare a blocchi, è chiaro che l'insieme degli autovalori di LH, è una combinazione degli insiemi di autovalori di $L_t C_{t+\tau} A_{t+1}^{t+\tau-1} B_t$, con t che viene definita come $t = \{0, \dots, N-\tau\}$.
Inoltre, anche Γ_k è una matrice diagonale a blocchi e di conseguenza anche la variabile Λ_k per come è stata definita, sarà una matrice triangolare a blocchi con gli autovalori che fanno parte della diagonale stessa della matrice.
Nello specifico, gli autovalori di Λ_k sono o uguali agli autovalori di **LH** oppure sono uguali a zero a seconda se il valore assunto della variabile $\gamma_k(\tau)$ è uguale a 1 oppure a zero.

3.3.2 Convergenze dei risultati

Da notare come per ogni $\gamma_k(t)$, abbiamo due possibilità di valori che corrispondono rispettivamente al pacchetto trasmesso (1) o al pacchetto perso (0).
Così facendo la matrice Γ_k ha $k \triangleq 2^{N+1-\tau}$ possibili risultati dovuti dall'indipendenza della variabile $\gamma_k(t)\ per\ ogni\ istante\ t-esimo\ di\ tempo.$
Come conseguenza la nuova matrice creata $\Lambda_k = \Gamma_k LH$ avrà a sua volta k possibili risultati.
Denotiamo l'insieme di tutte queste soluzioni come $\varrho = \{\Lambda^{(1)}, \dots, \Lambda^{(k)}\}$ e scendendo un po' più nel dettaglio, denotiamo $\Lambda^{(1)} = LH$ e $\Lambda^{(k)} = 0_{(N+1-\tau)}$ e corrispondono rispettivamente al caso in cui tutti gli $\gamma_k(t)$ sono uguali a 1 e 0.
Le altre *k-2* alternative sono anch'esse blocchi di matrici triangolari similari a **LH** ma con l'aggiunta di una o più righe di blocchi e identificano l'istante di tempo in cui il pacchetto è perso durante la trasmissione.
In altre parole, per una matrice Λ_k, se il pacchetto dati all'istante *t* è perso durante la trasmissione, $t \geq \tau$, poi la *(t+1- τ)esima* riga del blocco di Λ_k sarà una riga di zeri.

3.3.3 Condizioni di progettazione della matrice di guadagno d'apprendimento

Diamo ora le condizioni preliminari per progettare correttamente la matrice di guadagno d'apprendimento L_t definita nel paragrafo 3.3 con $0 \leq t \leq N - \tau$.
Per assicurare la corretta convergenza della legge di aggiornamento *P-type* del punto 3.26, la matrice di guadagno L_t dovrebbe soddisfare la *matrice di Hurwitz,* per cui una matrice quadrata ***M,*** viene definita appartenente alla famiglia delle matrici di Hurwitz se e solo se tutti gli autovalori di M sono a parte reale negativa.

$$-L_t C_{t+\tau} A_{t+1}^{t+\tau-1} B_t \qquad (3.29)$$

Dal *teorema di Lyapunov,* per ogni matrice **S** negativa definita con un'appropriata dimensione, ci sarà una matrice positiva **Q** tale che: $-((LH)^T Q + QLH) = \boldsymbol{S}$.
Nella seguente analisi, per facilitare i conti poniamo **S** = -I; Quindi esisterà una matrice **Q** per cui:

$$((LH)^T Q + QLH) = I \qquad (3.30)$$

Notare la differenza tra $\Lambda^{(i)}$ e LH, abbiamo che:

$$\left(\left(\Lambda^{(i)}\right)^T Q + Q\Lambda^{(i)}\right) \geq 0 \quad \text{per } i = 2,..,k-1 \qquad (3.31)$$

Quando consideriamo un modello di perdita di dati come per esempio, il *random sequence model,* non possiamo accedere a nessuna proprietà statistica per la perdita di dati. D'altronde l'ipotesi di lunghezza limitata della successiva perdita, ci assicura un modo deterministico per l'analisi di convergenza.

3.3.4 Considerazioni aggiuntive

In questi ultimi paragrafi, abbiamo preso in considerazione un sistema con l'assenza di rumore stocastico.
Proprio per questo motivo, abbiamo riscontrato che la convergenza della sequenza input ci assicura che l'output $y_k(t)$ possa tracciare in maniera estremamente precisa la traiettoria desiderata di riferimento $y_d(t)$.
Inoltre, la legge di aggiornamento utilizzata nel punto 3.24 utilizzata per questo tipo di sistema, si è rivelata estremamente utile poiché ci ha assicurato un'esponenziale velocità di convergenza dell'intero sistema.

3.4 Caso di un sistema stocastico

In questa sezione, consideriamo un sistema stocastico lineare presentato precedente quando abbiamo descritto la formulazione di un sistema MIMO.

$$\begin{aligned} x_k(t+1) &= A_t x_k(t) + B_t u_k(t) + w_k(t+1) \qquad (3.12) \\ y_k(t) &= C_t x_k(t) + v_k(t) \end{aligned}$$

Data l'esistenza dei rumori stocastici per ***sistemi reali***, la legge di aggiornamento 3.24 non può più garantire una convergenza stabile della sequenza dell'input.
Prendiamo in considerazione quindi la legge modificata 3.27; Se abbiamo una sequenza input U_k allora esisterà una limitazione di convergenza stabile.
È per questo che l'errore di tracciamento E_k è diviso in due parti, $\mathbf{H}(\boldsymbol{U_d} - \boldsymbol{U_k})$ e ξ_k.Così facendo è impossibile derivare il $lim_{k\to\infty} E_k = 0$, in caso contrario andremmo in contraddizione con l'assunto del punto 3.19.

In altre parole, il rumore stocastico non potrà mai essere eliminato da nessun algoritmo e quantomeno possiamo effettuare una previsione sull'effettivo andamento di tale tracciamento.
Dobbiamo imporre allora un meccanismo addizionale per ridurre gli effetti del rumore lungo l'asse di iterazione.
Nel concreto, per come l'abbiamo definito nel capitolo precedente, un appropriato guadagno decrescente per il termine di correzione nei processi di aggiornamento, risulta un requisito necessario e fondamentale per garantire la convergenza nel calcolo ricorsivo per l'ottimizzazione, identificazione e tracciamento del sistema stocastico.

3.4.1 Legge di aggiornamento per sistemi reali stocastici

Attraverso la legge di aggiornamento del punto 3.26, prendiamo tale legge e la modifichiamo rimpiazzando il parametro ρ con un'opportuna sequenza decrescente per far fronte a tali rumori.
Nello specifico la legge di aggiornamento dell'ILC per sistemi stocastici sarà come segue:

$$u_{k+1}(t) = u_k(t) + a_k \gamma_k(t+\tau) L_t e_k(t+\tau) \qquad (3.32)$$

Nel particolare, la sequenza decrescente è rappresentata proprio da $a_k \in 0{,}1$. Per come è stato dimostrato da molteplici risultati nel controllo e nell'ottimizzazione stocastica, l'introduzione di questa variabile rallenterà la velocità di convergenza dell'intero sistema.
Questo fatto è dovuto all'effetto di soppressione di a_k non solo imposto dal rumore stocastico in sé, ma anche dalle informazioni di correzione. Infatti, quest'ultimo è un classico compromesso tra una convergenza stabile a zero errori ed una velocità di convergenza ottimale per un controllo stocastico.
Quel che è giusto aspettarci, la velocità esponenziale di convergenza non sarà più garantita ma possiamo unicamente assicurarci una convergenza asintotica per un sistema stocastico.
In maniera similare al caso senza rumore trattato nei paragrafi 3.3, alziamo l'ingresso lungo l'asse dei tempi, così facendo la legge di aggiornamento la possiamo riscrivere come segue:

$$U_{k+1} = U_k + a_k \Gamma_k L E_k \qquad (3.33)$$

Nello specifico, $\boldsymbol{\Gamma_k}$ e $\mathbf{L}$ sono state già ben definite nel paragrafo 3.3. Giocando un po' con le variabili viste sino ad ora, sottraendo ambo i membri di un fattore U_d, sostituendo la definizione data in precedenza di $E_k = H(U_d - U_k) - \xi_k$ (3.19) e usando la notazione $\delta U_k = (U_d - U_k)$, arriviamo a dire che

$$\delta U_{k+1} = (I - a_k \Lambda_k)\delta U_k + a_k \xi_k \qquad (3.34)$$

Come detto in precedenza, $\Lambda_k = \Gamma_k \boldsymbol{LH}$, dobbiamo precisare che la matrice di guadagno L_t è la stessa utilizzata per lo studio del sistema senza rumore.
Di conseguenza la matrice L_t dovrà soddisfare $-L_t C_{t+\tau} A_{t+1}^{t+\tau-1} B_t$ per la *matrice di Hurwitz.*

3.5 Considerazioni fine capitolo

In questo capitolo, abbiamo considerato un ILC per *sistemi lineari* con perdita di dati, descritti attraverso il modello randomico RSM. In altre parole, nessuna proprietà statistica è stata imposta per i sistemi a perdita di dati randomici.
Quello che richiediamo da tale modello, è che il pacchetto ad un istante di tempo arbitrario, venga trasmesso correttamente almeno una volta per *K* iterazioni successive.
Per come è stato descritto e modellato, questo modello non può essere formulato sulla base del modello BVM (Bernoulli variable model).
Quindi, attraverso questo schema di dropout visto sino ad ora, utilizzeremo le leggi di aggiornamento ad intermittenza per stabilire e ricevere i risultati di convergenza il più fedeli possibili.
Inoltre, la convergenza quadratica media è anche assicurata per i sistemi MIMO con un grado arbitrario relativo studiato nel paragrafo 3.2 di questo capitolo.
Nel prossimo capitolo illustreremo la corrispondente derivazione per sistemi non lineari.

Capitolo 4: Modelli randomici per sistemi non lineari

In questa sezione ci concentriamo sul caso in cui la perdita di dati si verifichi solo sul lato output. Come per il capitolo 3, assumiamo che il canale di comunicazione lato attuatore funzioni bene senza nessun tipo di problema.
In tal caso, il problema principale al quale dovremmo prestare sufficiente attenzione è utilizzare tecniche di analisi adeguate per i vari modelli di dropout.
Nel particolare utilizzeremo i modelli studiati nel capitolo uno per sistemi non lineari con la presenza di rumore solo lato misurazione.

4.1 Formulazione del problema

Consideriamo il seguente sistema non lineare con la presenza di rumore lato misurazione:

$$x_k(t+1) = f\big(t, x_k(t)\big) + \boldsymbol{b}\big(t, x_k(t)\big)u_k(t) \qquad (4.1)$$
$$y_k(t) = \boldsymbol{c}(t)x_k(t) + v_k(t)$$

Nel particolare, $t \in \{0,1,\dots,\mathrm{N}\}$ denota l'istante di tempo del processo di un singolo processo mentre con k = {1, 2, …} identifichiamo la singola iterazione.
Le variabili $u_k(\mathrm{t}) \in R$, $x_k(t) \in R^n$ e $y_k(t) \in R$ denotano rispettivamente l'input, lo stato del sistema e l'output mentre $f\big(t, x_k(t)\big)$ e $\boldsymbol{b}\big(t, x_k(t)\big)$ sono funzioni continue e $\boldsymbol{c}(t)$ denota il coefficiente riferito all'output.
La variabile $v_k(t)$ è il nostro solito rumore stocastico che abbiamo largamente incontrato nei precedenti capitoli.
Per come è stato impostato il sistema 4.1, il segnale di misura viene ritrasmesso attraverso un *canale lossy.*
Il verificarsi di perdite di dati può essere assimilato ad uno switch che funziona in maniera randomica con stati di apertura e chiusura. Denotiamo M_k, un insieme di posizioni temporali in cui le misurazioni vengono perse nella k-esima iterazione. Quindi, $t_0 \in M_k$ se $y_k(t_0)$ è perso. L'obiettivo di controllo è quello di trovare una sequenza di controllo $\{u_k(t), k = 0,1,2,\dots\} \subset U$ in ambienti con forti perdite di dati per ridurre al minimo l'*indice medio di tracciamento*, tale per cui:

$$V(t) = \lim_{n\to\infty} sup\ 1/n \sum_{k=1}^{n} \big|\big|y_k(t) - y_d(t)\big|\big|^2 \qquad (4.2)$$

Inoltre, definiamo nella seguente maniera U come l'insieme dei possibili controlli ammessi:

$$U = u_{k+1}(t) \in \Im_k, sup\left||u_k(t)|\right| < \infty \qquad (4.3)$$

4.1.1 Schema a blocchi con perdita di dati

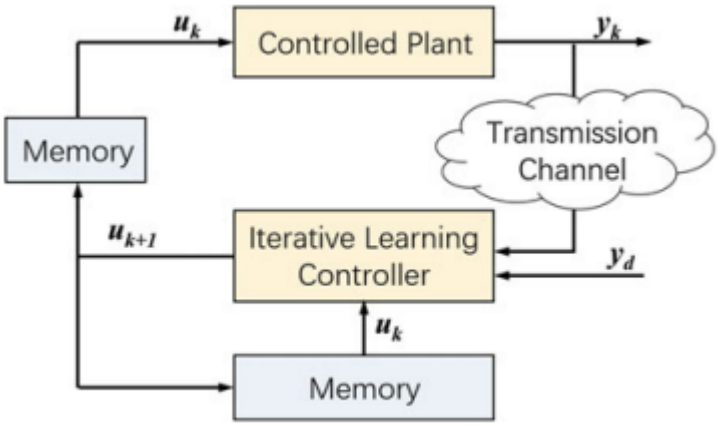

Figura 4.1: Sistema a blocchi di una rete con perdita di dati

Il modello base del sistema che prendiamo in esame è il medesimo già incontrato quando abbiamo parlato di sistemi lineari con perdite di dati.
Identifichiamo $y_d(t), t \in 0,1,\dots,\mathrm{N}$ come l'obiettivo di tracciamento. Quest'ultimo è realizzabile dal momento in cui esistono $u_d(t)$ e $x_d(0)$ per cui riscriviamo il sistema visto nel punto 3.1 nella seguente maniera:

$$x_d(t+1) = f\big(t, x_d(t)\big) + \boldsymbol{b}\big(t, x_d(t)\big)u_d(t) \qquad (4.4)$$
$$y_d(t) = \boldsymbol{c}(t)x_d(t)$$

Assumiamo che le funzioni $f(\cdot,\cdot)$ e $\boldsymbol{b}(\cdot,\cdot)$ siano continue tenendo in considerazione il secondo argomento.
Il numero reale $\boldsymbol{c}(t+1)\boldsymbol{b}(t,x)$ che accoppia l'input con l'output è sconosciuto ma il suo segno caratterizza la direzione del controllo ed è assunto come conosciuto.
Scendendo un po' più nel dettaglio, possiamo assumere $\boldsymbol{c}(t+1)\boldsymbol{b}(t,x) > 0$
Il valore iniziale può essere asintoticamente resettato ponendo gli stati iniziali e finali a zero.

$$x_k(0) \to x_d(0) \quad \forall k \to \infty \tag{4.5}$$

Per ogni istante di tempo *t,* la misurazione della perdita di dati è randomica senza obbedire a nessuna distribuzione di probabilità. Tuttavia, esisterà un numero K abbastanza grande tale che durante qualsiasi iterazione K successiva, almeno in una iterazione la misurazione verrà rispedita con successo.
Specifichiamo come il valore di K non deve esser conosciuto a priori. In altre parole, è richiesta solo l'esistenza di questa variabile.
Questa condizione specifica che le misurazioni non dovrebbero andare perse in maniera ingente per garantire la convergenza del sistema stesso.
Nel pratico, $K-1$ rappresenta il caso peggiore di una iterazione successiva n-esima con perdita di dati lungo l'asse di iterazione.

4.2 Schema di aggiornamento a intermittenza

In questo paragrafo vediamo lo schema di aggiornamento a intermittenza per i sistemi non lineari (4.1):

$$u_{k+1}(t) = u_k(t) + a_k \mathbf{1}_{(t+1)\notin M_k} \times \big(y_d(t+1) - y_k(t+1)\big) \tag{4.6}$$

Per ogni istante di tempo fissato dal punto 4.6 abbiamo che:

$$\delta u_{k+1}(t) = \delta u_k(t) - a_k \mathbf{1}_{(t+1)\notin M_k}[\boldsymbol{c}^+ \boldsymbol{b}_k(t)\delta u_k(t) + \varphi_{\mathrm{k}}(\mathrm{t}) - \mathrm{v_k}(\mathrm{t}+1)] \tag{4.7}$$

Dove $\varphi_k(t) = \boldsymbol{c}^+ \delta f_k(t) + \boldsymbol{c}^+ \delta \boldsymbol{b_k}(\boldsymbol{t})\boldsymbol{u_d}(\boldsymbol{t})$ (4.8)

Data l'esistenza delle funzioni non lineari $f_k(t)$ e $\boldsymbol{b_k}(\boldsymbol{t})$, risulta difficile formulare un vettore affine per un sistema non lineare (4.1) nella stessa maniera per come l'abbiamo definito nel sistema lineare.
Per questo motivo le tecniche utilizzate nel capitolo 3 non possono essere utilizzate in questo tipo di sistema ma si impiegheranno degli algoritmi complessi.
L'algoritmo che si utilizza per sistemi non lineari non funzionerà correttamente se il corrispondente pacchetto di misurazione andrà perso.
In altre parole, se i dati di tracciamento sono disponibili, l'algoritmo utilizza tali dati per l'aggiornamento; In caso contrario, se le informazioni di tracciamento andranno perse, l'algoritmo fermerà l'aggiornamento fino al nuovo arrivo di un nuovo pacchetto dati.

Questo meccanismo è semplice quanto efficace per contrastare le perdite di quest'ultime. Tuttavia, come si può osservare in applicazioni pratiche, se il pacchetto corrispondente viene perso nella k-esima iterazione, l'ultimo pacchetto disponibile allo stesso istante di tempo dell'iterazione precedente, potrebbe essere utilizzato per l'aggiornamento. In questa sezione siamo interessati nel vedere come si comporta l'algoritmo se l'ultimo pacchetto disponibile viene utilizzato per la successiva iterazione.

4.2.1 Errore di tracciamento

Per far ciò, sia $e_k(t) \triangleq y_d(t) - y_k(t)$ l'errore di tracciamento e utilizzando la formulazione 4.7 posso riscrivere l'iterazione successiva come:

$$e_k(t+1) = \boldsymbol{c}^{+}\boldsymbol{b}_k(t)\delta u_k(t) + \varphi_{\mathrm{k}}(\mathrm{t}) - \mathrm{v_k}(\mathrm{t}+1) \qquad (4.9)$$

Quindi, definiamo l'errore di aggiornamento utilizzato nella legge di aggiornamento dell'ILC nella seguente maniera:

$$e_k^*(t) = \begin{cases} e_k(t), & t \notin M_k \\ e_{k-1}^*, altrimenti \end{cases} \qquad (4.10)$$

Va sottolineato che la perdita di dati può avvenire in maniera randomica nelle iterazioni successive e per questo, l'errore di tracciamento $e_k^*(t)$, è definito ricorsivamente lungo l'asse di iterazione.
In altre parole, un errore di tracciamento $e_k(t)$ può essere utilizzato per diverse iterazioni successive se si verificano interruzioni sequenziali di dati.
Per questo, la legge di aggiornamento della sequenza di controllo è descritta come segue:

$$u_{k+1}(t) = u_k(t) + a_k e_k^*(t+1) \qquad (4.11)$$

In questo caso a_k ha un guadagno decrescente tale che $a_k > 0$, $a_k \to 0$, $\sum_{k=0}^{\infty} a_k = \infty$ oppure $\sum_{k=0}^{\infty} a_k^2 < \infty$. È chiaro come $1/k+1$ soddisfa tali requisiti.
Per un istante di tempo fissato, avremo che:

$$u_{k+1}(t) = u_k(t) + a_k \mathbf{1}_{(t+1)\notin \boldsymbol{M_k}} \boldsymbol{e_k}(\boldsymbol{t+1}) + \boldsymbol{a_k} \mathbf{1}_{(t+1)\in \boldsymbol{M_k}} \boldsymbol{e_{k-1}^*}(\boldsymbol{t+1}) \qquad (4.12)$$

L'evento 1_{evento} è un indicatore che vale 1 se l'evento indicato tra le parentesi graffe si verifica, varrà zero altrimenti.

4.2.2 Relativi risultati di convergenza

Per quando riguardano i risultati di convergenza riprendiamo il sistema 4.1 e l'indice 4.2, prendiamo in considerazione la *sequenza di controllo* $u_k(t)$ Generato dallo schema di aggiornamento dell'ILC proposto nel punto 4.10.
In questo modo, ci aspettiamo che $u_k(t)$ convergerà a $u_d(t)$ fino a quando l'iterazione k tenderà a infinito.
Per provare ciò, dimostriamo il tutto per induzione prendendo come riferimento l'asse temporale t e consideriamo per iniziare il caso per $t = 0$. Si è visto che nel punto 4.12, il segnale di controllo si aggiorna con le ultime informazioni di tracciamento dell'errore. Nello specifico, l'informazione di aggiornamento proviene proprio dall'ultima iterazione se non è avvenuta nessuna perdita di dati e, in caso contrario, dall'iterazione precedente ma non adiacente.
Per essere più chiari, introduciamo una sequenza temporale di arresto $\tau_0 < \tau_1 < \tau_2 \dots$ per indicare tutte le iterazioni in cui le misure all'istante di tempo $t = 1$, trasmetteranno con successo; Avremo quindi che $1 \in M_k$.
Entrando più nel dettaglio, poniamo $\tau_0 = 1$ e per quanto detto nel paragrafo precedente avremo che $\tau_i - \tau_{i-1} < K$
Vedendo la figura sottostante 4.2, l'errore di traiettoria verrebbe trasmesso correttamente alla τ_i- esima iterazione mentre si perderà nelle restanti iterazioni.
Come risultato l'aggiornamento dell'errore $e_k^*(1)$ verrà identificato come $e_{\tau_i}(1)$.
Da quanto detto sino ad ora e con l'aiuto dei punti 4.10 e 4.12 avremo che:

$$u_{\tau_i+1}(0) = u_{\tau_i}(0) + a_{\tau_i} e_{\tau_i}(1) \qquad (4.13)$$

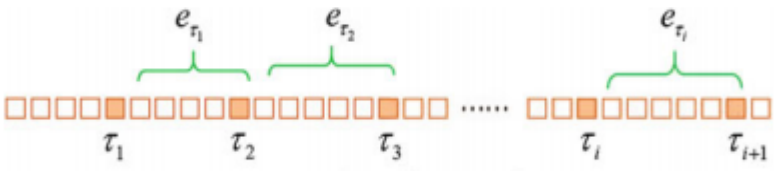

Figura 4.2

4.3 Perdita di dati ambo i lati

Fino ad ora abbiamo studiato i casi in cui la perdita di dati, affliggeva solo il lato misurazione poiché era più facile studiarlo e perché non produceva asincronia all'interno del sistema.
In questo momento cercheremo di vedere brevemente i casi in cui la perdita di dati, affligge sia il lato misurazione che lato attuatore cercando di capire le conseguenze che ne comporta.

4.3.1 Formulazione del problema

Consideriamo come al solito un sistema similare a quello già visto nel punto 4.1:

$$x_k(t+1) = f\big(t, x_k(t)\big) + \boldsymbol{B}(t)u_k(t) \qquad (4.14)$$
$$y_k(t) = \boldsymbol{C}(t)x_k(t)$$

All'inizio del capitolo abbiamo già definito in maniera accurata ogni singola variabile del sistema; Invece, per le variabili **C** e **B** sappiamo solo che sono sconosciute con dimensioni appropriate. Per semplificare possiamo dire che $C^+B(t) \triangleq C(t+1)B(t)$.
Se prendiamo la traiettoria di riferimento $y_d(t)$, lo stato iniziale $x_d(0)$ può essere riscritto tale che: $y_d(0) = C(0)x_d(0)$.
Tengo a specificare che esisterà sempre un input desiderato $u_d(t)$ che genererà il segnale di riferimento $y_d(t)$.
Nel dettaglio l'input desiderato $u_d(t)$ può essere definito in maniera ricorsiva nella seguente maniera:

$$u_d(t) = \Big[\big(C^+B(t)\big)^T C^+B(t)\Big]^{-1}\big(C^+B(t)\big)^T \times \Big(y_d(t+1) - C(t+1)f\big(t, x_d(t)\big)\Big) \qquad (4.15)$$

Con questo segnale di controllo, è evidente che le seguenti equazioni per il desiderato riferimento sono soddisfatte. In altre parole, l'input $u_d(t)$ calcolato nel punto 4.15, spinge l'impianto a generare il riferimento desiderato $\mathrm{y_d(t)}$.

4.3.2 Schema con perdita lato attuatore e misuratore

Consideriamo una formulazione generale di un ILC in cui il plant e il controllore sono connessi da una rete cablata o wireless come nella figura sottostante 4.3.
In questo sistema esistono due reti che vanno dal plant al controllore di apprendimento denominato *lato misurazione* e dal controllore d'apprendimento al plant chiamato *lato attuatore.*
Entrambi le reti soffrono di perdita di dati randomici e per modellare quanto detto, introduciamo due nuove variabili $\sigma_k(t)$ e $\gamma_k(t)$ soggette al modello di distribuzione di Bernoulli per entrambi i lati.
In altre parole, entrambi $\sigma_k(t)$ e $\gamma_k(t)$ varranno 1 se il pacchetto sarà trasmesso correttamente, 0 altrimenti. Notiamo anche come entrambi le reti lavorano individualmente e di conseguenza è razionale presumere che $\sigma_k(t)$ sia indipendente da $\gamma_k(t)$ e viceversa.

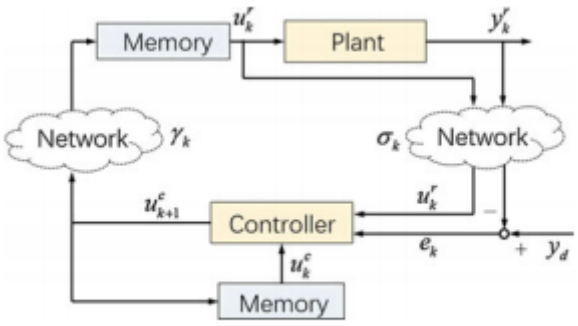

Figura 4.3

L'obiettivo di controllo di questo paragrafo è quello di progettare un adeguato schema di aggiornamento tale che la sequenza di input generata, garantisca una convergenza di zero errori.
Inoltre, l'output del sistema può tracciare il riferimento desiderato, in maniera asintotica fino a quando il numero di iterazioni tenderà a infinito.
Per raggiungere questo obiettivo di controllo, la legge di aggiornamento segue una strategia di base. Per essere precisi, se i dati vengono trasmessi correttamente ***lato misurazione***, allora il controllore d'apprendimento aggiornerà il suo segnale d'ingresso. Altrimenti se i dati vengono persi durante la trasmissione sempre sul lato misurazione, il controllore interromperà l'aggiornamento e conserverà il segnale d'ingresso precedente.
D'altra parte, se il segnale input verrà trasmesso con successo ***lato attuatore,*** allora il plant utilizzerà il segnale d'ingresso appena arrivato. In caso contrario se il segnale d'ingresso verrà perso durante la trasmissione, il plant manterrà il segnale input memorizzato della precedente iterazione.

Per rendere il tutto più conciso, definiamo l'input generato dal controllore d'apprendimento come un segnale d'ingresso ***calcolato*** denotato da $u_k^c(t)$ mentre identifichiamo l'input utilizzato dal plant come un segnale d'ingresso ***reale*** indicato da $u_k^r(t)$; Di seguito formuliamo il segnale input aggiornato all'iterazione $k+1$.

$$u_{k+1}^c(t) = \sigma_{k+1}(t)u_k^r(t) + [1 - \sigma_{k+1}(t)]u_k^c + \sigma_{k+1}(t)L_t e_k(t+1) \quad (4.16)$$

Identifichiamo L_k la matrice di guadagno. Inoltre, il segnale input reale usato dal plant viene definito come:

$$u_{k+1}^r(t) = \gamma_{k+1}(t)u_{k+1}^c(t) + [1 - \gamma_{k+1}(t)]u_k^r(t) \quad (4.17)$$

Si noti come le perdite di dati randomiche si verificano indipendentemente su entrambi i lati di misurazione e attuatore, quindi l'aggiornamento dei segnali sia reali che calcolati potrebbero essere asincroni.
Proprio per questo il segnale input calcolato, viene aggiornato quando i dati sono ritrasmessi con successo dal plant.
Tuttavia, l'ultimo segnale input potrebbe non essere ritrasmesso al plant in modo tale che il segnale d'ingresso reale mantenga l'informazione all'istante precedente.
In tal caso, si verifica l'asincronia tra i due tipi di segnali d'ingresso, tra quelli reali e calcolati.
Inoltre, vale la pena sottolineare che anche l'aggiornamento di entrambi gli input sarà asincrono in riferimento all'asse dei tempi poiché le variabili $\sigma_k(t)$ e $\gamma_k(t)$ sono indipendenti dai vari istanti temporali.
Va anche notato che lo schema ILC fornito nella figura 4.3, non richiede alcun problema di crescita transitoria quando si verificano perdite di segnali output. Questo è dato perché quando si verificano grandi errori temporali dati dall'asincronia del sistema, il controllore può avere nessuna informazione su di loro e di conseguenza non possono fermare la crescita transitoria.

4.4 Considerazioni aggiuntive

In questo capitolo abbiamo affrontato il problema dell'ILC per sistemi non lineari con la presenza di rumori che inficiavano sulle prestazioni che andavano dall'output al controllore con perdita di dati. Utilizzando il modello RSM, sia per l'intermittenza che per l'aggiornamento dello schema delle istruzioni successive, è stata provata la convergenza del sistema.
Tuttavia, per arrivare ai risultati trovati, è stata necessaria la conoscenza preliminare della direzione del controllo, in altre parole dovevamo conoscere il segno del valore di accoppiamento tra l'input e l'output.

Capitolo 5: Applicazione pratica e relativo codice Matlab

In questo capitolo cercheremo di riprendere quanto detto negli scorsi quattro capitoli e vedremo l'applicazione pratica con i relativi grafici per sistemi ILC (con e senza perdita di dati) che ho avuto come riscontro durante il periodo di tirocinio.
Nel particolare, considereremo d'ora in poi un sistema lineare di secondo ordine con un'eventuale perdita di dati solo *lato misurazione,* modellata dal modello *BVM.*

5.1 Algoritmo Base del sistema

Per facilitarci il problema, lo studio è stato effettuato su un sistema lineare di tipo *SISO* a tempo variante, la *traiettoria di riferimento* presa in esame è del tipo:

```
y_d = sin(2*pi*traiettoria);        (5.1)
```

Nello specifico il prodotto per la *traiettoria* è rappresentato da un array per la simulazione in riferimento ai tempi del sistema t_s e ai tempi di simulazione unitari $T = 1$; In altre parole ho:

```
traiettoria = (0:t_s:T);            (5.2)
```

Qui sotto metterò un breve estratto di codice per analizzare il corretto funzionamento dell'algoritmo al passaggio di ogni iterazione:

```
1    for t = index
2
3          x_k(:,t) = A*x_k(:,(t-1)) + B*u_k(t-1) + w_k(t);
4          y(t) = C*x_k(:,t) + v_k(t-1);
5
6          e_k(t) = y_d(t) - y(t);
7
8          u_k_nuova(k,t-1) = u_k(t-1) + guadagno * e_k(t) * m_k(k,t);
9
10    end
11
12   u_k = u_nuova;
```

Estratto di codice di implementazione
di un sistama ILC (5.3)

Il codice 5.3 mostrato è la rappresentazione *generale* del funzionamento di un modello ILC con presenza di errori stocastici ed errori di misurazione tramite Matlab.
Per simulare la perdita, ho fissato il numero di iterazioni $k = 10$; Le variabili A, B e C sono state opportunamente formulate come sistemi matriciali con appropriate dimensioni.
Nella prima parte di codice, nello specifico nelle righe 3, 4 e 6 abbiamo l'evoluzione degli stati, dell'output e dell'input.
Nel particolare, $x_k(t)$ rappresenta proprio lo stato del sistema modellato principalmente dall'evoluzione degli stati delle precedenti iterazioni.
Tengo a precisare che per lo stato iniziale, u_k avrà le sembianze proprio di y_d descritta nel punto 5.1.
$e_k(t)$ è proprio l'errore di tracciamento dato dalla differenza tra la traiettoria di riferimento y_d e l'output y_k.
Tengo a precisare che nel caso dei risultati *senza* perdite di dati, i parametri M_k, w_k e v_k non saranno ovviamente presenti in quanto rappresentano esclusivamente interesse nel caso con dropout.

5.1.2 Legge di aggiornamento

La legge di aggiornamento u_k mostrata nella riga 8 è stata modellata seguendo la legge vista per la prima volta nel *capitolo 1* e prende il nome di *P-type ILC update law.*
Nell'estratto di codice, quest'ultima è stata denominata "*u_k_nuova"* poiché rappresenta proprio la legge che aggiorna gli stati iniziali del sistema lungo l'evoluzione dell'asse temporale *t.*
Notiamo anche come l'errore è moltiplicato per un *guadagno (*che abbiamo opportunatamente chiamato *guadagno d'apprendimento*).
Inoltre, il termine "$guadagno * e_k(t+1)$" (riga 8) è proprio il *termine di innovazione* visto nel capitolo 1.2.3.
In aggiunta, vediamo come il termine d'innovazione è moltiplicato per la variabile M_k che rappresenta l'insieme randomico delle posizioni temporali in cui le misurazioni vengono perse nella k-esima iterazione.
Per finire, possiamo notare che alla fine di tutta l'evoluzione del sistema, u_k si aggiornerà al passaggio di ogni iterazione (riga 12).

5.2 Risultati grafici

Qui di seguito riporto i vari risultati dei vari grafici che ho avuto utilizzando il modello descritto all'inizio di questo capitolo.
Nel particolare andremo a suddividere due diverse tipologie di risultati; La convergenza dell'output senza alcuna presenza di perdite di dati e la convergenza dell'output con la presenza randomica in riferimento, nel particolare, anche della stringa di codice che abbiamo visto poco fa.

5.2.1 Istanti iterativi e traiettoria di riferimento

In questo paragrafo ci soffermiamo sul carattere della traiettoria di riferimento y_d (definita in blu scuro) e sull'output raggiunto ad ogni ciclo iterativo attraverso l'implementazione dell'algoritmo visto in parte nel punto 5.3.
Possiamo notare come, col passare degli istanti iterativi, l'input prenderà le sembianze della *traiettoria di riferimento.* Infatti, la traiettoria blu chiara rappresenta la prima iterazione, fino ad arrivare alle traiettorie rosso scuro che rappresentano le ultime iterazioni.
La fedeltà nel replicare la reference trajectory dipenderà principalmente dalla matrice di guadagno che gli abbiamo passato nella *legge di aggiornamento.* Si è notato come al diminuire del guadagno, le iterazioni finali si discostavano sempre di più dalla traiettoria presa di riferimento.
Questo ci fa capire come, nel sistema senza la presenza di perdite di dati, la fedeltà dell'output dipenderà principalmente dal valoro assunto dal guadagno.

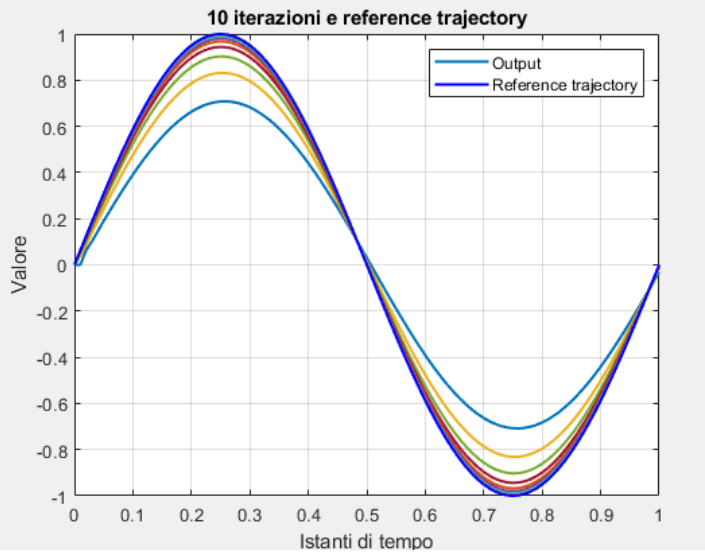

Grafico 5.4

Per valori molto piccoli avremo che il nostro output si discosterà in maniera pronunciata dalla traiettoria y_d come visto nel seguente grafico.

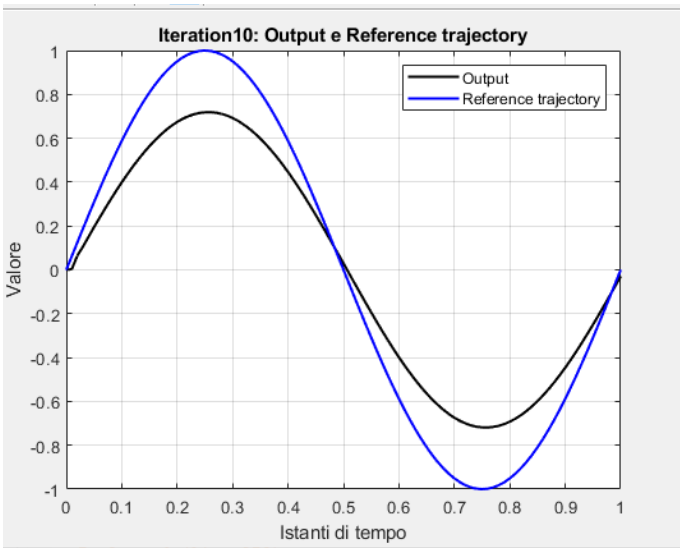

Grafico 5.5

5.2.2 Errori di tracciamento

Vorrei un attimo soffermarmi sui risultati avuti per quanto riguardano gli errori di tracciamento per il solito sistema che non presenta alcuna perdita di dati.
Nel grafico 5.6 sottostante, vediamo come col passare degli istanti di tempo, il valore di e_k tende a zero.
Ciò è il risultato atteso perché da quanto appreso nei primi capitoli, quando l'errore e_k è diverso da zero, l'input u_k non è ottimo e di conseguenza toccherà iterarlo nelle successive k-esime iterazioni in modo tale da migliorare le prestazioni del sistema.
In caso contrario,u_k sarà ottimale e non serviranno ulteriori iterazioni poiché l'errore tenderà ad un valore più quanto prossimo allo zero.
Questo specifico miglioramento è una funzione di combinazioni lineari formati da u_k ed e_k, nello specifico l'input $u_k + 1$ per la k+1esima iterazione verrà generato prima dell'esecuzione della k+1esima iterazione e subito dopo verrà inviato al *plant* per l'iterazione successiva.

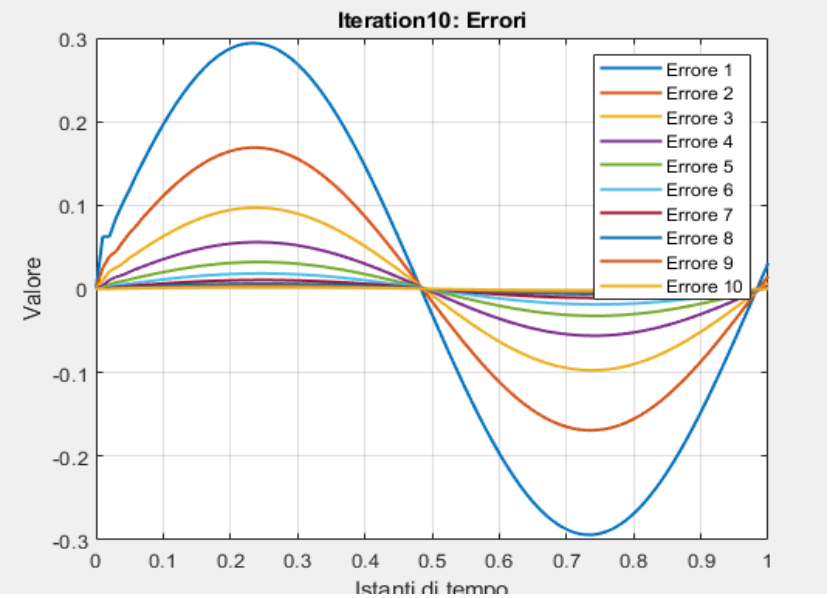

Grafico 5.6

Vediamo come per gli errori $e_k(1)$ ed $e_k(2)$, abbiamo valori che si discostano in maniera più accentuata dallo zero perché rappresentano proprio le prime iterazioni in cui il sistema deve ancora analizzare al meglio il plant.
Negli istanti iterativi finali (segnati dai colori arancione e giallo) abbiamo e_k prossimo allo 0, in accordo a quanto detto poco fa.
Per renderlo ancora più conciso e facile da vedere, ho realizzato il grafico 5.7 che rappresenta l'insieme di tutti gli errori che tenderanno a zero col passare delle varie iterazioni partendo da un valore più alto nel primo istante iterativo, in relazione a quanto detto poco fa.

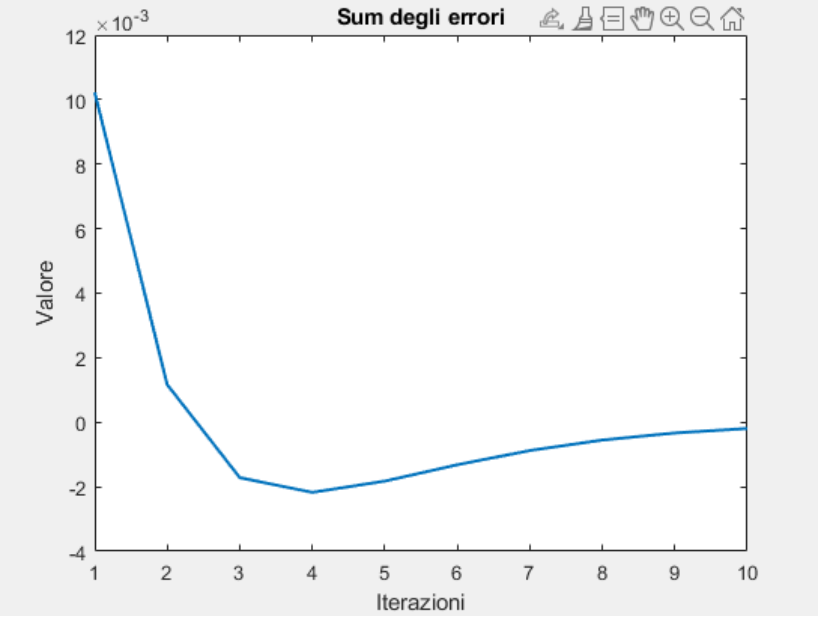

Grafico 5.7

5.3 Risultati con perdite di dati

In questa sezione, ci soffermeremo sull'analizzare i risultati grafici sul sistema ILC con la presenza di perdita di dati randomici.
Il codice visto nel punto 5.3 lo utilizzeremo sempre come base di partenza e per analizzare questi risultati, vorrei prima soffermarmi sul come è stato possibile realizzarli.
Per prima cosa è stata opportunatamente modificata la covarianza dei fattori randomici $v_k(t)$ e $w_k(t)$ per poter avere anche a livello grafico dei risultati plausibili in istanti iterativi ben definiti.
Un altro elemento fondamentale è stato generalizzare una nuova variabile denominata semplicemente *p.*
Quest'ultima è definita come *parametro di successo* ed è fondamentale per poter studiare i fattori randomici.

$$p \in \{0,1\} \qquad (5.8)$$

In altre parole, questa variabile assumerà valori compresi tra 0 e 1, nel particolare 1 rappresenterà la probabilità di successo massima per trasmettere informazioni con nessuna perdita di dati. Valori più vicini allo zero rappresenteranno invece, probabilità di successo minori per trasmettere informazioni senza perdita di dati; Nel nostro caso ho inizializzato $p = 0.7$.

```
m_k = (rand(iterazioni,N)<p);
```
(5.9)

Nell'estratto di codice 5.9 possiamo vedere come ho utilizzato p. In questo caso viene implementato per poter definire proprio M_k; In altre parole, rappresenterà l'insieme randomico delle *N* posizioni temporali della k-esima iterazione e, per valori minori del *parametro di successo,* il sistema perderà o trasmetterà correttamente l'informazione di quella specifica iterazione. A livello grafico ci dovremmo aspettare dei spike che identificano proprio questi cali prestazionali.

5.3.1 Istanti iterativi e traiettoria di riferimento

In questo paragrafo ci soffermiamo sul carattere della traiettoria di riferimento y_d (definita in blu scuro) e sull'output raggiunto ad ogni ciclo iterativo attraverso l'implementazione dell'algoritmo visto in parte nel punto 5.3 e utilizzando i parametri aggiuntivi descritti poco fa.
Per descriverli nella maniera più accurata, inizierei a vedere i grafici, sempre in riferimento alla *traiettoria di riferimento*, in istanti ben definiti per non creare caos nel grafico stesso.

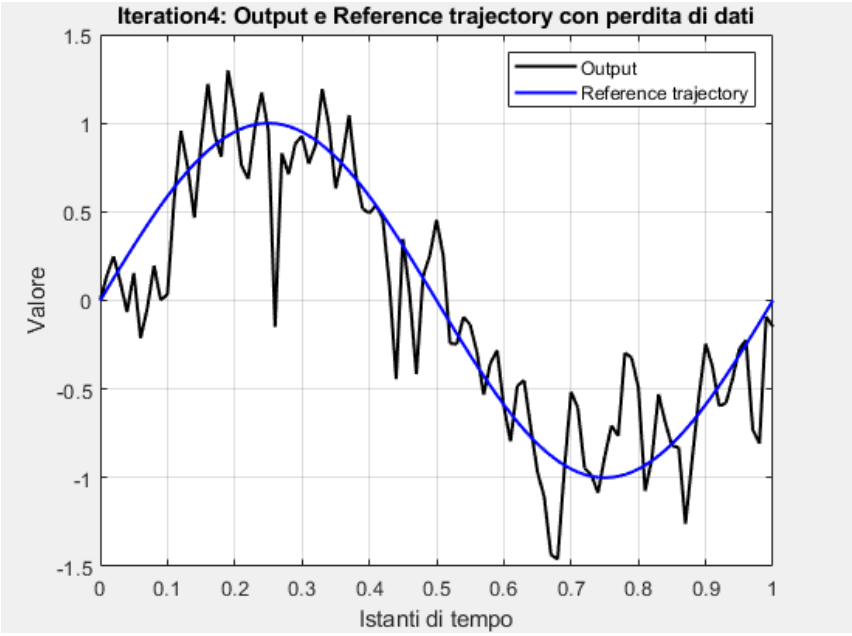

Grafico 5.10

Il grafico 5.10 identifica proprio uno specifico istante iterativo (precisamente il quarto) in cui le perdite di dati sono ancora molto accentuate, possiamo comunque notare come l'output y_k prenda le sembianze della *reference trajectory (*linea blu) fin dai primi istanti iterativi ma non avendo una conoscenza ancora completa del plant, gli errori (raffigurati come detto prima dai spike) sono molteplici e inficiano pesantemente nelle prestazioni generali dell'intero sistema.
Nello specifico possiamo definire gli *spike* come gli istanti precisi in cui avvengono le incorrette trasmissioni con la conseguenza presenza di perdite di dati.

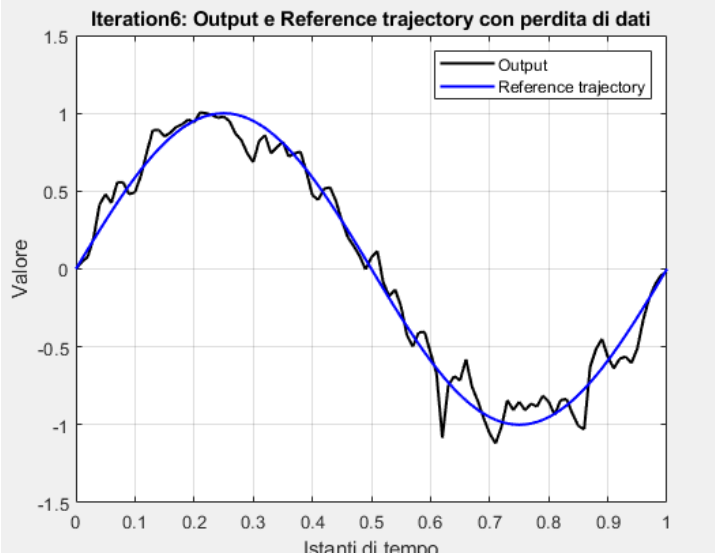

Grafico 5.11

Notiamo come già dalla sesta iterazione (5.11), l'output prenda più precisamente le sembianze della traiettoria di riferimento.
Gli *spike* sono meno frequenti e meno invasivi, sinonimo di un buon controllo d'apprendimento da parte di ILC e di una maggior conoscenza dell'impianto da parte dell'algoritmo.

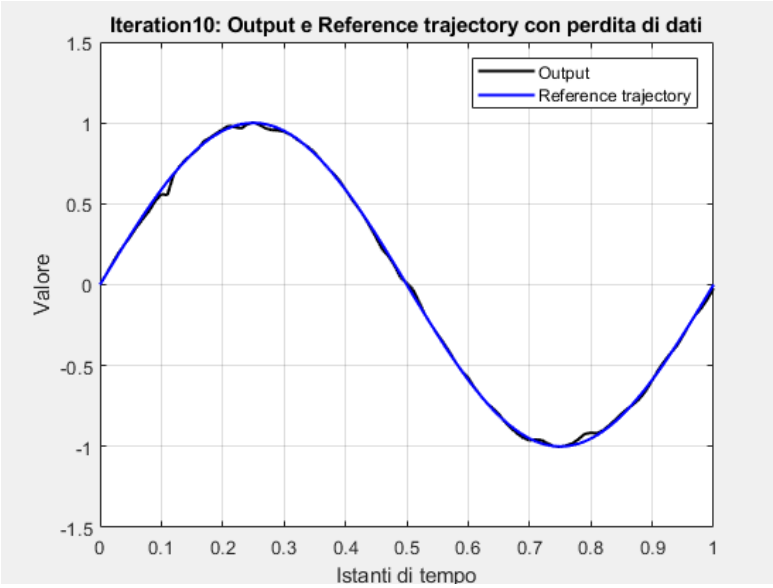

Grafico 5.12

Per finire, vediamo come nella decima ed ultima iterazione y_k ha preso completamente le sembianze della *reference.* Attraverso dieci iterazioni si è giunti ad un valore aspettato dell'output attraverso l'utilizzo di un corretto algoritmo di apprendimento.
Vediamo come gli errori di traiettoria ed i suoi conseguenti spike, sono stati quasi completamente annullati, sinonimo di un buon approccio alla perdita di dati randomica nei vari istanti iterativi.

5.3.2 Errori di tracciamento con dropout

In questa sezione, vedremo alcuni errori di tracciamento riscontrati durante le fasi di test dell'algoritmo.
Nel particolare, vedremo attraverso dei grafici il modo in cui ILC cerca di approssimare e compensare le perdite di dati dell'iterazione precedente attraverso i soliti algoritmi visti in questo capitolo.
Nel grafico 5.13 possiamo vedere la linea rossa che delimita i dieci istanti iterativi presi in esame. Scendendo nel dettaglio vediamo come ho preso in esame le ultime due iterazioni, in modo tale da avere due traiettorie più quanto simili, avendo così una conoscenza del plant maggiore, rispetto alle prime fasi.
Denotiamo la traiettoria blu come il nono istante iterativo con una perdita di dati randomica presente principalmente tra gli istanti di tempo 0.2, 0.3 e 0.8.
La traiettoria gialla rappresenta la decima ed ultima iterazione e possiamo vedere come ILC cerchi di compensare quelle mancanze dell'iterazione precedente (date dalla perdita *randomica* di dati) dove possibile. Ricordo che lo scopo finale del sistema è di ricreare un output il più simile alla traiettoria di riferimento y_d.

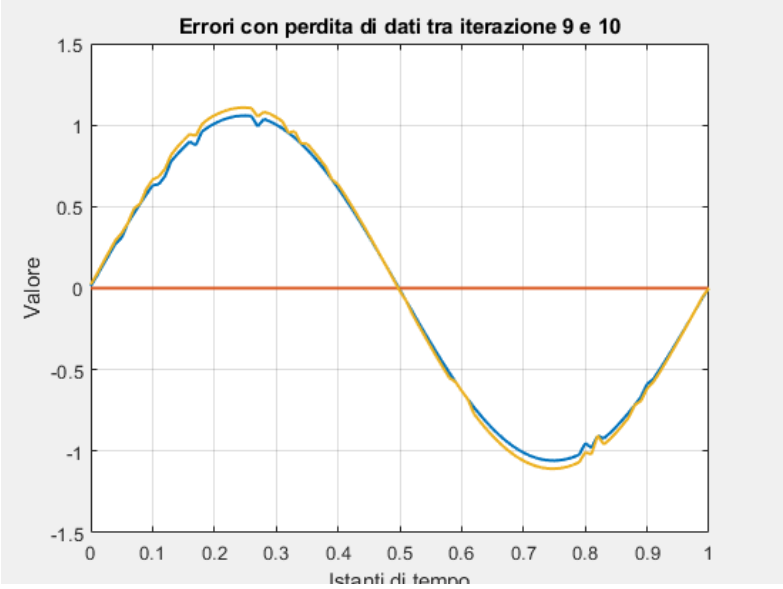

Grafico 5.13

5.3.3 Errori di tracciamento con dropout pt.2

Per vedere la correttezza dell'algoritmo in maniera definitiva, ho cercato di cambiare alcuni parametri, tra cui la *reference trajectory,* il guadagno ed i sistemi matriciali A, B e C per vedere se l'algoritmo continuasse a convergere.

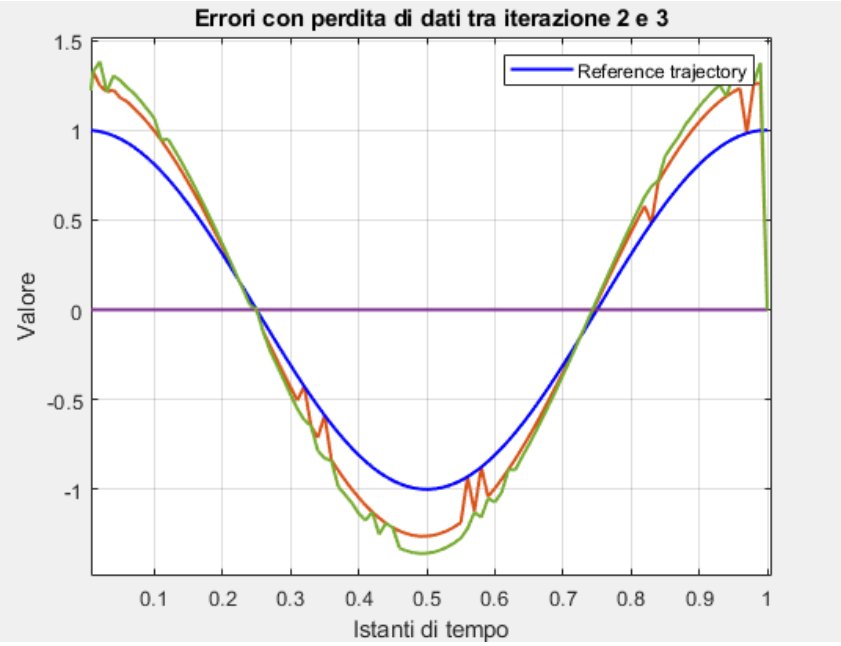

Grafico 5.14

Prendendo questa volta le prime iterazioni come soggetto d'esame, si può vedere come l'algoritmo cerchi sempre di riprendere una traiettoria il più quanto simile alla traiettoria di riferimento (identificata col colore blu scuro).
Stando nelle prime iterazioni, vediamo come l'output si discosti in maniera più accentuata da y_d; Nel particolare identifichiamo l'output arancione come il secondo istante iterativo mentre il verde identificherà di conseguenza il terzo istante.
Anche in questo caso l'algoritmo cerca di compensare le perdite di informazione dell'istante iterativo precedente in modo tale da assottigliare i problemi dovuti alle perdite randomiche con la traiettoria di riferimento.
Notiamo come nell'istante di tempo finale, entrambi gli output tenderanno a zero, fatto che ci saremmo dovuti aspettare per com'è stato definito l'errore di traiettoria.

Capitolo 6: Conclusioni

Arrivati a questo punto, dopo lo studio dell'algoritmo ad apprendimento iterativo, siamo arrivati a capire il suo funzionamento e soprattutto, la sua utilità nell'ambito principalmente industriale ma non solo. Abbiamo visto come un sistema del genere possa essere usato, per esempio, anche in ambiti medici e biomedici per l'aiuto alla deambulazione di un paziente o per l'assistenza ventricolare a livello cardiaco.
Abbiamo visto come l'obiettivo di ILC, è quello di arrivare al miglior risultato possibile anche con la presenza di fattori esterni che ne inficiano le sue prestazioni.
Lo scopo di fornire un tracciamento perfetto rimarrà sempre l'obiettivo primario e garantirà ad ILC l'assicura affidabilità se quest'ultimo sarà progettato a dovere attraverso la combinazione dei dati forniti dal sistema e dalle proprie conoscenze pregresse.
Questo studio fornisce una base per il raggiungimento degli obiettivi che si prefigge ILC, dando tutti gli strumenti necessari per uno sviluppo futuro sempre più promettente di tali modelli.

Indice delle Figure

Bibliografia

Magni L., Scattolini R.: *Complementi di controlli automatici.* Nel particolare studio del filtro di Kalman (2006).

Xu, J.X, Tan, Y.:*Linear ant NonLinear Iterative learning control*. Springer, New York (2003)

Ahn, H-S., Moore, K.L., Chen, Y.Q.: *Iterative learning control: Robustness and monotonic convergence for interval System.* Berlino (2007).

Liu, T., Gao, F.: ILC- based iterative learning control for batch process with time delay variation. J. Process control (2010)

Ahn, H.S., Chen, Y.Q., Moore, K.L.: Interminttent iterative learning control: Proceedings of the 2006 IEEE International Symposium on Intelligent Control (2006).

Huang, S.N., Tan, K.K., Lee, T.H.: Necessary and sufficient condition for convergence of iterative learning algorithm (2002)

[1] Pan, Y.-J., Marquez, H.J., Chen, T., Sheng, L.: Effects of network communications on a class of learning controlled non-linear systems. Int. J. Syst. Sci. 40(7), 757–767 (2009)

[2] Shen, D., Wang, Y.: Iterative learning control for networked stochastic systems with random packet losses. Int. J. Control 88(5), 959–968 (2015)

[3] Shen, D., Wang, Y.: ILC for networked nonlinear systems with unknown control direction through random Lossy channel. Syst. Control Lett. 77, 30–39 (2015)

[4] Liu, J., Ruan, X.: Networked iterative learning control approach for nonlinear systems with random communication delay. Int. J. Syst. Sci. 47(16), 3960–3969 (2016)

[5] Shen, D., Chen, H.-F.: Iterative learning control for large scale nonlinear systems with observation noise. Automatica 48, 577–582 (2012)

[6] Seel, T., Schauer, T., Raisch, J.: Iterative learning control for variable pass length systems. In: Proceedings of the 18th IFAC world congress, pp. 4880–4885 (2011)

Ringraziamenti finali: