ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA

SEDE DI CESENA

Seconda Facoltà di Ingegneria con Sede a Cesena

Corso di Laurea Magistrale in

INGEGNERIA ELETTRONICA E TELECOMUNICAZIONI PER LO SVILUPPO SOSTENIBILE

UN AMBIENTE DI DIAGNOSTICA PER MACCHINE AUTOMATICHE

Tesi in

SISTEMI E TECNOLOGIE PER L'AUTOMAZIONE LM

Relatore

Prof. Ing. Carlo Rossi

Correlatore

Ing. Matteo De Angeli

Presentata da

Marco Vecchi

Sessione III

Anno accademico 2011/2012

“Se ascolto dimentico,
se vedo ricordo,
se faccio capisco”

Confucio

Indice

Elenco delle Figure

Ringraziamenti

Desidero innanzitutto ringraziare il relatore della mia tesi, il professor Carlo Rossi, per avermi dato la possibilità di contribuire allo sviluppo del progetto di *start-up* denominato *Optima*, come attività di tirocinio finalizzato alla tesi in azienda. La frequentazione del corso "Sistemi e Tecnologie per l'Automazione LM", presieduto dal professore, ha consentito di effettuare il piccolo, ma non semplice, salto culturale che divide l'ingegneria dell'informazione dall'ingegneria industriale.

Ringrazio l'ingegnere Matteo De Angeli, correlatore della tesi e ideatore del progetto, per la fiducia dimostrata nei miei confronti, per il pieno coinvolgimento nella nuova attività e per la disponibilità nel trasmettermi alcune competenze informatiche, utili nell'attività.

Ringrazio il consorzio *Spinner* per aver creduto nella bontà del progetto e, attraverso il programma della Regione Emilia-Romagna per lo sviluppo di idee e progetti innovativi, per aver reso possibile lo sviluppo del progetto stesso, finanziando con una borsa di studio la mia attività di ricerca.

Figura 1: Logo Spinner2013.

Ringrazio infine l'azienda *Tissue Machinery Company*, TMC, con la quale si è instaurato un prezioso rapporto di collaborazione. L'interesse e la piena disponibilità, sin da subito dimostrate, hanno reso possibile la fase di sperimentazione presso la loro sede bolognese.

Introduzione

Per poter garantire un alto livello di prestazioni, sicurezza e affidabilità nelle macchine automatiche è importante che guasti di componenti ed errori di sistema siano rilevati in modo tempestivo, o addirittura predittivo, e che la causa e la gravità di ciascun malfunzionamento siano diagnosticate in modo da poter intraprendere le opportune azioni di correzione. La tempestività, inoltre, consentirebbe alle aziende un'importante diminuzione dei fermi macchina e quindi ad evidenti risparmi. La diagnosi di tipo predittiva potrebbe permettere una riduzione dei danni provocati da guasti o malfunzionamenti, o in alcuni casi perfino la loro prevenzione.

Da questi concetti prende spunto il progetto di ricerca denominato *Optima*, iniziato nel 2011 presso il DEIS, Dipartimento di Elettronica, Informatica e Sistemistica dell'Università di Bologna, con l'obiettivo di studiare con un approccio formale un software diagnostico, configurabile e *general purpose*, per macchine automatiche e macchine utensili.

E' all'interno di questo contesto che si colloca la tesi da me svolta in azienda. La mia attività prevedeva l'interfacciamento con le macchine automatiche e la ricerca di algoritmi diagnostici da affiancare ad un prodotto software sviluppato durante un precedente lavoro di ricerca universitaria finalizzato a rilevare eventuali guasti, utilizzando correlazioni logiche tra i vari segnali della macchina, quali allarmi e segnali provenienti da sensori. Attraverso un linguaggio brevettabile, tale software è dunque in grado di risalire alla causa del guasto, a fronte di segnali provenienti dal PLC, in maniera deterministica. In parallelo, grazie al mio *background* culturale più orientato all'ingegneria dell'informazione, è stato possibile fornire uno strumento diagnostico, *general purpose*, orientato all'analisi delle preziose informazioni contenute nei segnali della macchina automatica.

Gli obiettivi della mia tesi possono essere riassunti in due attività principali:

- interfacciamento fra la macchina automatica e il sistema diagnostico
- ricerca ed implementazione di algoritmi per una diagnosi di tipo predittiva

La tesi è suddivisa, così come l'attività svolta in azienda, in due parti fondamentali: una prima parte di ricerca, concentrata sull'aspetto teorico, ed una seconda di sperimentazione, improntata sullo studio di protocolli, l'implementazione degli algoritmi e la valutazione dei risultati ottenuti.

Il primo capitolo introduce i concetti fondamentali dell'ambito diagnostico presentando la terminologia di base, necessaria alla sua comprensione. Verrà presentata, in particolare, la fondamentale attività di *Fault Detection and Isolation*, FDI. L'attività FDI può essere inquadrata in due metodi principali: un primo approccio basato sul modello del sistema, *model-based*, ed uno basato sui segnali che fanno parte del sistema stesso, *signal-based*.

In questa tesi si è analizzato, in particolare, il potente strumento delle reti neurali artificiali, inquadrabile come approccio di tipo *model-based*. Nel capitolo 2 si presenta tale strumento: le origini, il funzionamento e le possibili applicazioni.

Il capitolo 3 espone alcuni metodi utili all'attività di *Fault Detection and Isolation* secondo l'approccio *signal-based*.

Per quanto riguarda la seconda parte della tesi, nel capitolo 4, dopo una necessaria introduzione della macchina automatica e dei diversi sistemi che la compongono, si è presentata la macchina messa a nostra disposizione da TMC per la fase di sperimentazione, la FTS-300. Si tratta di una macchina impacchettatrice per rotoli di carta, in grado di raggiungere la velocità di 200 pacchi al minuto.

Il capitolo 5 espone gli standard utilizzati e le caratteristiche del primo modulo software realizzato, denominato *data-logger*. Il suo compito è quello di interfacciarsi con la macchina automatica, monitorando determinati segnali della macchina stessa, e di gestire l'archiviazione dei relativi dati.

Il capitolo 6 mostra la fase di test degli algoritmi e degli standard studiati. In particolare vengono presentate le simulazioni di guasto provocate sulla macchina, i dati raccolti nei diversi test effettuati e le relative valutazioni.

Il capitolo 7 espone le conclusioni riguardo l'ambiente diagnostico progettato, propone possibili approcci algoritmici per il rilevamento delle anomalie e presenta gli sviluppi futuri e le possibili prospettive.

Le appendici, infine, presentano il codice realizzato per implementare i diversi algoritmi analizzati.

Capitolo 1:

Fault Detection and Isolation

Per poter addentrarsi nel mondo della diagnostica è opportuno introdurre alcuni concetti preliminari.

In Figura 2 è rappresentato un generico sistema con input $\boldsymbol{i}(t) \in R^m$ e output $\mathbf{o}(t) \in R^d$. Il sistema è affetto da un *fault,* o malfunzionamento, rappresentato da *f*.

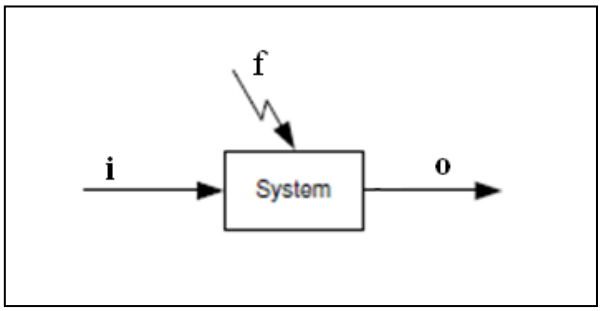

Figura 2: Sistema con ingressi **i**, uscite **o**, affetto da un *fault* f.

Il *fault* può essere interpretato come un evento indesiderato che porta ad operazioni anomale nel sistema. Le modalità con cui tali guasti si ripercuotono sulle operazioni sono numerose e possono essere divise in tre gruppi:

- *fault* moltiplicativi $o_i^f = \rho \cdot o_i$
- *fault* additivi $o_i^f = o_i + \Delta o_i$
- *fault* intermittenti $o_i^f = o_i + \delta \cdot \Delta o_i$

dove Δo_i rappresenta un offset, ρ una costante moltiplicativa e δ l'impulso di Dirac.

L'evento indesiderato tenderà a degradare nel complesso le prestazioni, seppure questo non implichi obbligatoriamente *failure*, cioè avaria, di componenti fisici. Si utilizza il termine *fault* anziché *failure* per sottolineare la tollerabilità di tale

malfunzionamento; con il termine *failure*, infatti, si intende una fase successiva, in cui, a seguito di un ulteriore degrado, si verifica il collasso del sistema.

La capacità di un sistema di non subire *failure* a fronte di presenze di *fault* viene detta *fault tolerance*, tolleranza ai guasti [1].

Ci sono sei tipologie principali di guasti sui processi industriali:

1. guasti sui componenti del sistema;
2. guasti sulla strumentazione (sensori e attuatori);
3. guasti sui sistemi di controllo;
4. guasti sul sistema energetico;
5. disturbi e interferenze dall'esterno;
6. errori umani.

La funzione di *fault detection*, o rilevamento guasti, ha il compito di constatare se un *fault* $f \in F$ si è verificato nel sistema, dove F rappresenta il set di tutti i guasti che potrebbero verificarsi. Si tratta, molto semplicemente, di una semplice decisione binaria: o qualcosa non è andato a buon fine o tutto si comporta correttamente. Questa funzione, apparentemente banale, assume in realtà importanza strategica sia da un punto di vista economico sia ambientale, proprio perché in grado di evitare il collasso del sistema.

Dopo la prima fase di semplice rilevamento, è possibile affermare che *qualcosa* non funziona correttamente ma non è dato sapere *cosa,* in realtà, non funzioni correttamente. Per questo motivo in un'applicazione reale, la *fault detection* è seguita da un passo successivo, la funzione di *fault isolation*, o isolamento guasti. In questa fase si procede alla determinazione del modo guasto, *failure mode*, cioè quel fenomeno esterno che rende visibile il disturbo. In termini pratici viene individuato il *fault* f_i all'interno del set F di tutti i possibili guasti. Il processo combinato di *fault detection* e *fault isolation*, in letteratura, viene abbreviato con l'acronimo FDI [2].

In ultima analisi risulterà fondamentale associare i modi guasto verificatisi alle relative cause guasto. In questo modo l'utente sarà in grado di riconoscere a monte

da dove è partito il problema e prendere le opportune contromisure. L'attività di associazione prende il nome di *Failure Modes and Effects Analysis*, FMEA, e in Figura 3 si trova un esempio [3].

componente	Failure mode	Possibili cause	Effetti sul sistema
interruttore	•P.B. bloccato chiuso •Contatto P.B. bloccato chiuso	•Failure meccanica •Failure meccanica •Errore umano	•Motore attivo troppo a lungo→corto circuito → corrente elevata → fusibile che si apre
relè	•Contatto bloccato aperto •Contatto bloccato chiuso	•Failure meccanica •Failure meccanica •Corrente elevata attraverso contatto	•Motore fermo •Motore attivo troppo a lungo: corto circuito, corrente elevata e fusibile che non apre il circuito
Fusibile	Non si apre	•Failure primaria •Errore umano (scelta errata del fusibile)	In caso di corto, il fusibile non apre il circuito
Motore	•Fermo •Corto circuito	•Failure primaria •P.B. bloccato chiuso •Failure primaria •Motore attivo troppo a lungo	•Motore fermo •Corrente elevata, apertura fusibile, contatto relè bloccato chiuso

Figura 3: Esempio FMEA

Per rendere un sistema *fault tolerant* e quindi capace di non incorrere in *failure* in caso di guasti, occorre introdurre il concetto di ridondanza. In pratica viene sfruttata l'idea intrinseca di abbondanza. La ridondanza fisica, o hardware, consiste nella disposizione all'interno del sistema di un numero maggiore di componenti critici rispetto allo stretto necessario. Purtroppo tale approccio si può rivelare spesso inapplicabile; l'eccessivo costo, di fatto, la rende adottabile solo in alcuni casi. Rappresenta una prima tecnica utile al rilevamento di guasti o meglio al loro mascheramento. Infatti il guasto avviene ed il componente affetto da fault rimane attivo ma non viene percepito dall'utente finale.

Una soluzione più ingegneristica consiste nell'impiego della ridondanza analitica. Non ci si affida più alla disposizione di più copie di uno stesso componente critico, ma a relazioni matematiche opportune del sistema. Attraverso tali relazioni si potrà

mettere a confronto una certa caratteristica, relativa all'istante attuale, e la stessa caratteristica, ma in condizioni operative nominali. Uno scostamento fra tali valori evidenzierà la presenza di un comportamento anomalo.

Le relazioni matematiche possono essere classificate in due categorie principali, dando quindi vita a due diversi approcci:

- basato su modello o *model-based*: per decidere se è avvenuto un guasto viene usato un modello, che può essere matematico o può essere basato su qualche conoscenza del sistema.

- basato su segnale o *signal-based*: vengono effettuate elaborazioni di tipo statistico o matematico sulle misure acquisite.

In realtà una terza strada è stata intrapresa già a partire dagli anni '90, quella dell'intelligenza artificiale, cioè lo studio di sistemi in grado di svolgere funzioni e ragionamenti tipici della mente umana. Il cervello umano ha capacità di elaborazione e deduzione a dir poco straordinarie. Un ramo dell'intelligenza artificiale si è occupata della creazione di un modello matematico del neurone; da qui la nascita dei primi sistemi neurali artificiali.

Vi sono molte versioni discordanti riguardo la classificazione di tale approccio: chi vuole inquadrarla tra le *model-based*, chi tra le *signal-based*, chi in una nuova categoria *knowledge-based*. Come sempre, c'è un fondo di verità in ciascuna delle precedenti visioni. Quella che più si avvicina alla realtà, però, è l'approccio *model-based* poiché l'intelligenza artificiale va ad approssimare il modello del sistema attraverso l'analisi e il confronto di alcune variabili.

Capitolo 2:

Approccio *model-based*

Il classico approccio di tipo *model-based*, nell'attività di *Fault Detection and Isolation*, prevede una rigorosa trattazione matematica, attraverso la conoscenza del modello del sistema. Il metodo è molto potente ma al tempo stesso può essere applicato raramente. Infatti la conoscenza del modello matematico del sistema accade di rado, o nel caso di sistemi molto semplici. L'approccio delle reti neurali artificiali è una possibilità molto interessante, poiché il compito della conoscenza delle relazioni esistenti tra le variabili, all'interno di un sistema complesso, può essere demandato alla rete stessa.

2.1 Reti Neurali Artificiali

2.1.1 Il neurone

Le reti neurali artificiali prendono spunto dal principio di apprendimento del cervello umano. L'unità funzionale del sistema nervoso è il neurone e si stima che nella corteccia cerebrale ne siano presenti circa 125 miliardi.

I neuroni sono cellule nervose adibite alla creazione e allo scambio di segnali elettrici.

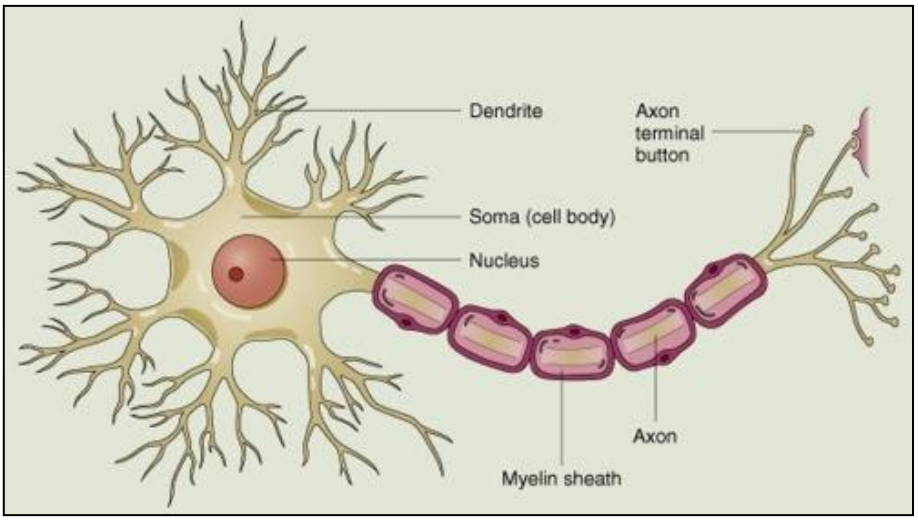

Figura 4: Rappresentazione schematica del neurone.

Dalla parte centrale del neurone, chiamata soma, hanno origine due tipologie di prolungamenti citoplasmatici, detti neuriti, che sono i dendriti e l'assone. I dendriti hanno ramificazioni come un albero, nelle diverse direzioni, per captare i segnali dall'esterno e propagarli in direzione centripeta. L'assone, invece, è deputato alla trasmissione centrifuga, verso altre cellule. A differenza dei dendriti, l'assone è un buon conduttore, anche per via del diametro più uniforme come si può notare in Figura 4. La parte terminale è detta bottone terminale ed è l'importante punto di contatto con i dendriti o il corpo cellulare di altri neuroni. In questo modo si forma il circuito neuronale consentendo la propagazione dell'impulso elettrico. Gli assoni sono ricoperti da due membrane protettive che fungono da isolante: la guaina di *Schwann* e la guaina mielinica. Periodicamente si verificano delle strozzature dette *nodi di Ranvier*, in corrispondenza dei quali si interrompe la guaina mielinica. Alla base della creazione degli impulsi elettrici, detti *spike,* vi è un semplice meccanismo di polarizzazione della membrana del neurone, di tipo ondulatorio [4] [5].

Il cervello umano è dunque una complessa rete di neuroni e la complessa fase di apprendimento altro non è che una modifica della propagazione del segnale in base alle esperienze. Da molto tempo la scienza cerca di fare piccoli passi alla scoperta di questo grande mistero che è il cervello. Già sono stati fatti passi importanti come l'individuazione delle diverse aree funzionali del cervello e la tracciabilità degli

impulsi celebrali, ma il campo della neurologia ha davanti a sé ancora un universo inesplorato; basti pensare alle complesse elaborazioni, alle emozioni, al processo di memorizzazione, etc. Le potenzialità del cervello umano sono sbalorditive e molto spesso sottostimate.

È da qui che prende spunto l'ingegneria per la realizzazione di sistemi artificiali in grado di automatizzare la risoluzione di quei problemi che il cervello risolve nella quotidianità: è la nascita delle *reti neurali artificiali*. Uno degli aspetti che ne sottolinea le potenzialità riguarda le innumerevoli applicazioni: dall'ambito finanziario a quello biomedico, dal campo industriale alle previsioni meteo.

2.1.2 Il percettrone

Il comportamento del neurone viene approssimato dal percettrone, il quale agisce come un processore e contiene la base di conoscenza tra le variabili dipendenti ed indipendenti del sistema.

Il percettrone riceve in input un vettore **i** che rappresenta i segnali raccolti dai dendriti, composto da n elementi (n dendriti). Poi effettua la combinazione lineare tra il vettore in input **i** e il vettore dei pesi **w** che rappresenta l'attenuazione introdotta dai singoli dendriti. Infine per la teoria della decisione si applica la funzione di attivazione *f*. In realtà ogni percettrone al suo interno, oltre che combinare linearmente i suoi input, sommerà un certo valore *b*, detto *bias*. Per semplificare il tutto però, conviene pensare al bias come ad un ulteriore ingresso di valore costante pari ad 1, a cui verrà applicato il peso *w* con valore pari a *b*.

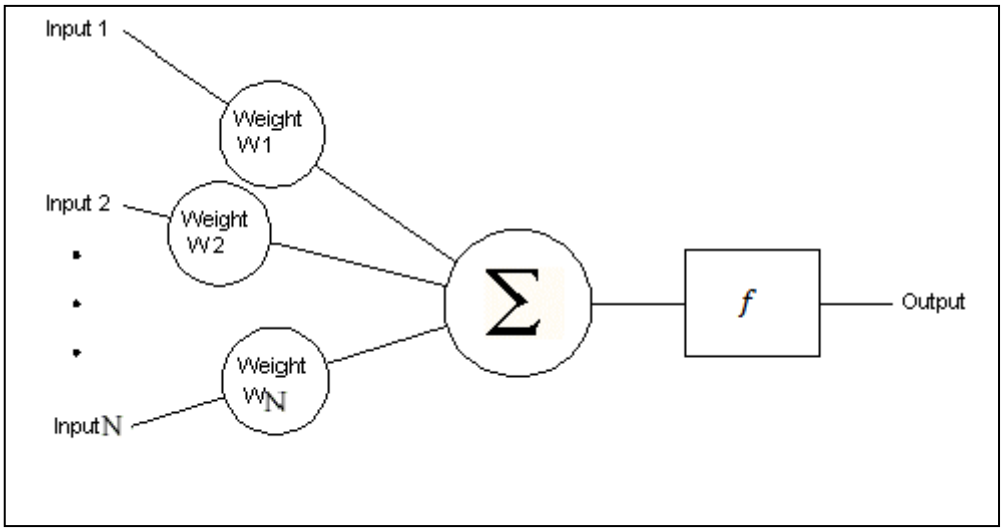

Figura 5: Rappresentazione del percettrone

Il percettrone risulta dunque caratterizzato dal seguente modello matematico:

$$o = f\left(\sum_{r=1}^{n} i_r \cdot w_r\right) \qquad 1.$$

dove o rappresenta l'output, i_r l'r-esimo input e w_r l'r-esimo peso.

Dunque i parametri che caratterizzano il percettrone sono il vettore dei pesi **w** e la funzione f. La funzione f viene scelta in fase di progettazione. In natura viene utilizzata una funzione gradino, o *Heaviside*: quando in input viene superata una soglia predefinita, il neurone propaga lo *spike*, altrimenti non accade nulla. Nelle implementazioni pratiche si utilizzano funzioni non lineari ma continue e differenziabili. La più utilizzata è la funzione sigmoidale.

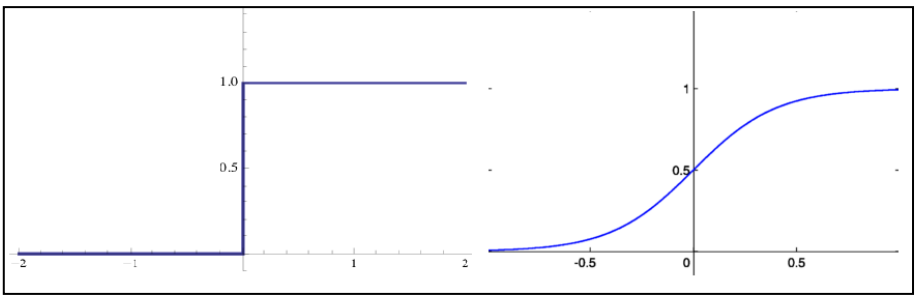

Figura 6: Funzione gradino e funzione sigmoidale.

I pesi **w** sono semplici coefficienti e costituiscono le interconnessioni tra i vari percettroni. Tutte le potenzialità della rete neurale risiedono in questi pesi, poiché racchiudono il *mapping* tra input e output. Un peso di valore "0" può essere assimilato ad una connessione aperta; uno di valore "1" produce in uscita un segnale invariato. I pesi **w** sono il risultato della minimizzazione della funzione di *Mean Square Error,* che quantifica l'errore commesso dalla rete:

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{j=1}^{m}\left(f\left(\sum_{r=1}^{n} w_r \cdot i_r^j\right) - t_j\right)^2 \qquad 2.$$

dove *r* rappresenta l'indice degli *n* input mentre *j* l'indice degli *m* esempi del *training*. Infine t_j rappresenta il risultato desiderato del percettrone, detto *target*, in corrispondenza del j-esimo esempio del *training set* in input.

2.1.3 Architettura di una rete neurale artificiale

Una rete neurale artificiale, o *Artificial Neural Network, ANN,* è costituita da un insieme di percettroni connessi in cascata ed organizzati in strati. L'architettura più classica è quella rappresentata in Figura 7 e prende il nome di percettrone multi strato, *multi-layer perceptron*. Lo strato più a sinistra è detto *input layer* e ha il semplice compito di fornire lo stesso input a tutti i percettroni dello strato centrale, detto *hidden layer.* Per questo motivo non è composto da percettroni ma da banali propagatori. Sono possibili architetture che prevedono più strati *hidden*, ma si è visto non essere conveniente da un punto di vista di *trade-off* tra prestazioni e complessità. Tutte le uscite dei percettroni dello strato intermedio saranno input dell'ultimo strato, noto come *output layer.* La struttura evidenziata è un esempio di *feed-forward*, cioè una rete interamente connessa, in cui i percettroni prendono gli input solo dallo strato precedente e forniscono gli output solo allo strato successivo. In questo modo si evita la presenza di rischiosi cicli.

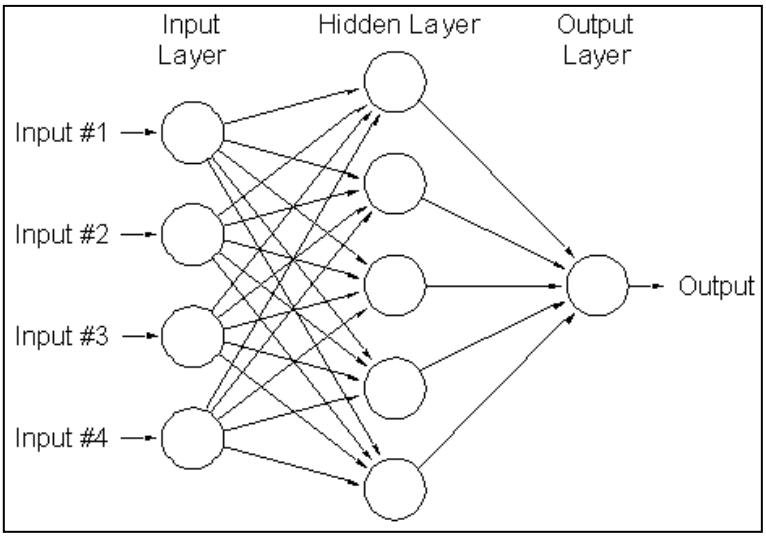

Figura 7: Architettura classica di una rete neurale.

È necessario ora un po' di formalismo per esprimere le relazioni fra *input* e *hidden layer*. Per evitare incomprensioni si farà uso d'ora in avanti del pedice *r* per l'indice dell'*input layer*, *i* per l'*hidden* e *h* per l'*output layer*:

$$q_i = f(c_i) \qquad 3.$$

$$dove\ c_i = \sum_{r=1}^{t} h_r^i \cdot i_r \qquad 4.$$

f indica sempre la funzione di attivazione, q_i rappresenta l'uscita dell'i-esimo percettrone dell'*hidden layer*, i_r l'r-esimo elemento dei dati di input, o *dataset*, h_r^i il peso da applicare all'r-esimo input riferito all'i-esimo percettrone dell'*hidden layer*.

La stessa relazione può essere espressa in forma matriciale:

$$\bar{c} = \bar{\bar{h}} \cdot \bar{\imath} \qquad 5.$$

Analogamente è possibile esprimere analiticamente le relazioni fra *hidden* e *output layer*:

$$o_h = f(d_h) \qquad 6.$$

$$dove\ d_h = \sum_{i=1}^{n} w_i^h \cdot q_i \qquad 7.$$

o_h rappresenta l'uscita dell'h-esimo percettrone dell'*output layer*, w_i^h il peso da applicare all'i-esimo percettrone dell'*hidden layer* riferito all'h-esimo percettrone dell'*output layer*.

In forma matriciale:

$$\bar{d} = \bar{\bar{w}} \cdot \bar{q} \qquad 8.$$

w_i^h e h_r^i saranno dunque i generici elementi di due matrici $\bar{\bar{w}}$ e $\bar{\bar{h}}$ che caratterizzano l'intera rete [6].

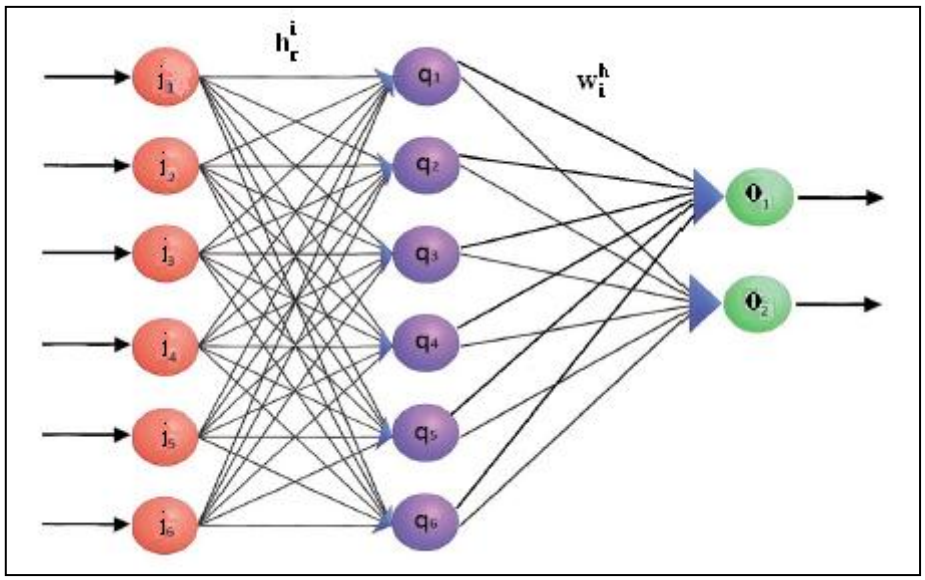

Figura 8: Matrici dei pesi

2.1.4 Algoritmo di *back-propagation*

Il punto di forza della rete neurale, come citato precedentemente, sta nella multidisciplinarietà delle sue applicazioni. Viene infatti adottata in contesti molto particolari quali il riconoscimento (audio, video, …), la classificazione, l'approssimazione di modelli e funzioni, predizioni, etc.

Sorge spontaneo un quesito: come rendere *special purpose* uno strumento intrinsecamente *general purpose*?

Attraverso la fase di addestramento.

Nella creazione di una rete vi sono infatti due fasi:

- *training* o addestramento.
- *test* o verifica.

Nella fase di *training* la rete è addestrata per catturare le relazioni esistenti tra input e output. Nella fase di *test* invece, si valutano le prestazioni della rete con una parte del *dataset* non utilizzata per il *training* [7]. Per addestrare la rete si adotta una tecnica denominata *algoritmo di back-propagation* che consiste sostanzialmente nella correzione dei pesi dei percettroni, propagando l'errore all'indietro.

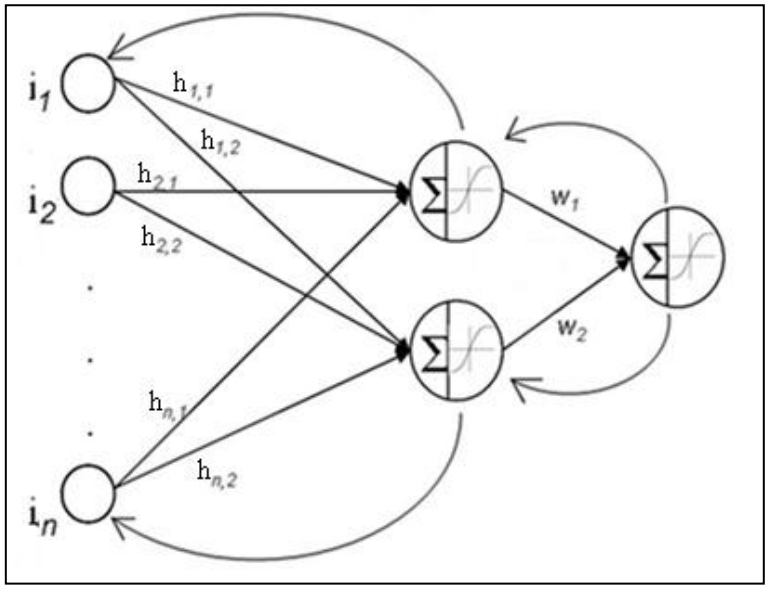

Figura 9: Algoritmo di back-propagation

Come anticipato nel paragrafo 2.1.2 l'obiettivo è la minimizzazione del *Mean Square Error* (MSE):

$$E(\boldsymbol{w}) = \frac{1}{p}\sum_{h=1}^{p}(o_h - t_h)^2 \qquad 9.$$

dove h rappresenta l'indice dei p percettroni di uscita, o indica l'uscita reale mentre t il target, cioè l'uscita desiderata.

Per la ricerca del punto di minimo della funzione di MSE si parte da valori casuali dei pesi per poi spostarsi sulla funzione stessa di un passo, detto *learning rate,* verso la direzione di discesa più ripida. Il metodo più semplice che implementa questa ricerca è detto *Gradient Descent,* che sfrutta il gradiente della funzione:

$$\nabla E = \left(\frac{\partial E}{\partial w_{1,1}}, \frac{\partial E}{\partial w_{1,2}}, \cdots, \frac{\partial E}{\partial w_{n,p}}\right) \qquad 10.$$

Il gradiente di una funzione indica la direzione di massima pendenza della stessa.

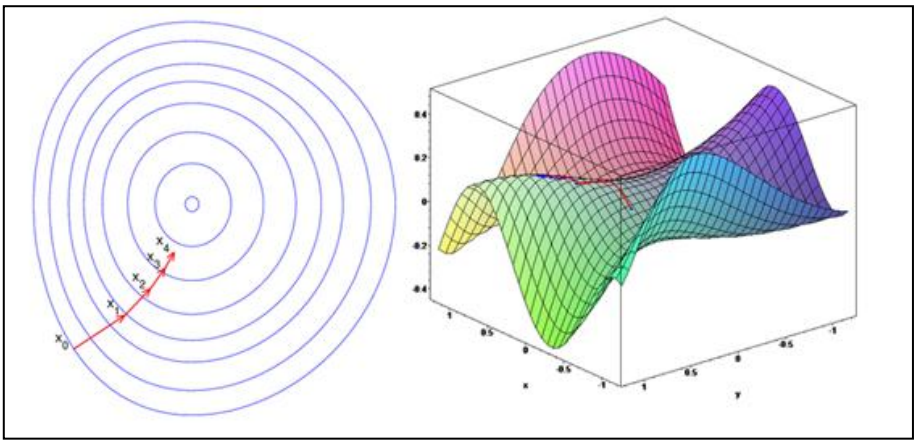

Figura 10: Due esempi di applicazione del Gradient Descent

L'idea è dunque quella di spostarsi lungo la direzione opposta individuata dal gradiente:

$$\Delta w_i^h = -\gamma \frac{\partial E}{\partial w_i^h} = \gamma q_i e_h \qquad 11.$$

$$w_i^h(n+1) = w_i^h(n) + \Delta w_i^h \qquad 12.$$

Per non appesantire in maniera eccessiva la trattazione è stata omessa la dimostrazione [8] dell'ultimo passaggio dell'equazione 11. Con la lettera *n* si è indicato l'istante temporale discretizzato; *n*+1 indicherà dunque l'istante futuro.

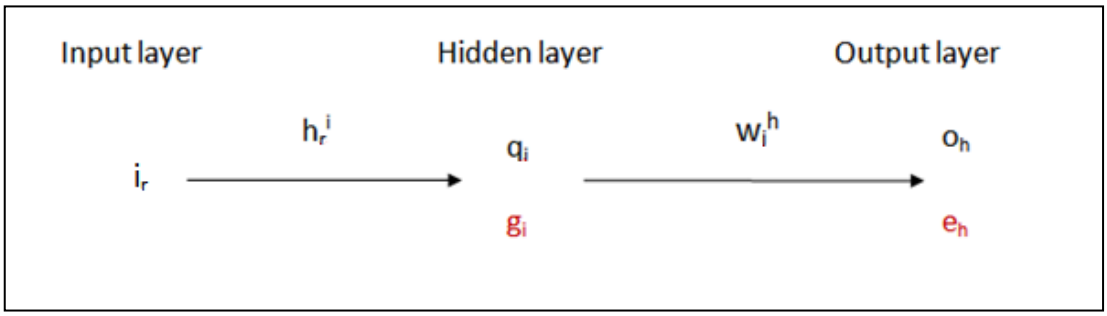

Figura 11: Schema concettuale della rete

Nelle formule, γ indica il passo di spostamento, detto *learning rate,* per la minimizzazione dell'errore. Un valore troppo elevato non consentirebbe il raggiungimento del minimo, troppo piccolo invece renderebbe eccessivamente lenta la convergenza. q_i indica l'uscita dell'i-esimo percettrone dell'*hidden layer*. e_i rappresenta una quantità assimilabile ad un errore.

Tale "errore"viene definito come:

$$e_h = f'(d_h) \cdot (o_h - t_h) \qquad 13.$$

Applicando gli stessi ragionamenti ai pesi dell'*hidden layer* e definendo in maniera opportuna g_i, si ottiene in maniera analoga:

$$\Delta h_r^i = -\gamma \frac{\partial E}{\partial h_r^i} = \gamma i_r g_i \qquad 14.$$

dove g_i viene definito come:

$$g_i = f'(c_i) \sum_{h=1}^{n} e_h w_i^h \qquad 15.$$

Dalla formula 15 si può osservare come g_i risulti essere una vera e propria retro-propagazione all'*hidden layer* dell'errore in *output layer*. È proprio da questa retro-propagazione dell'errore che nasce il nome dell'algoritmo.

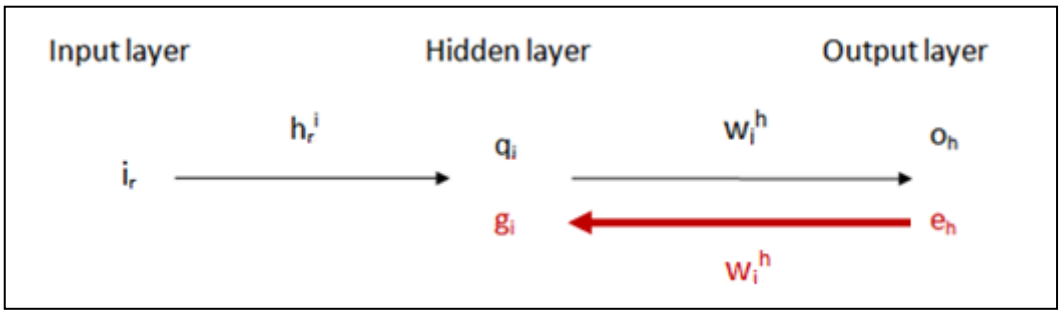

Figura 12: Back-propagation degli errori.

In conclusione l'aggiornamento delle due matrici. Dalla 12 si aggiorna quella riferita all'*output layer*:

$$\overline{w_h}(n+1) = \overline{w_h}(n) + \gamma \cdot e_h \cdot \bar{q}(n) \qquad 16.$$

Di conseguenza si aggiorna anche quella relativa all'*hidden layer*:

$$\bar{h_i}(n+1) = \bar{h_i}(n) + \gamma \cdot g_i \cdot \bar{\imath}(n) \qquad 17.$$

Grazie agli ultimi due passaggi, è stato effettuato uno spostamento sulla funzione E, in questo caso MSE, di un passo γ, in direzione opposta al suo gradiente, ∇E, calcolato sulla configurazione precedente dei pesi. In realtà la teoria dell'ottimizzazione ha dimostrato che l'utilizzo di un *learning rate* costante risulta poco efficiente. L'ideale sarebbe quello di ricalcolare ad ogni iterazione un nuovo valore del parametro γ attraverso un algoritmo denominato *line search*; tutto ciò chiaramente a discapito di una maggior complessità [9].

Per velocizzare la convergenza e reagire ad eventuali minimi locali si è introdotto un nuovo parametro μ, detto momento, che modifica l'aggiornamento dei pesi.

$$\Delta w_i^h(n+1) = -\gamma \frac{\partial E}{\partial w_i^h} + \mu \cdot \Delta w_i^h(n) \qquad 18.$$

Il nuovo parametro definisce l'importanza dell'aggiornamento del peso precedente nel nuovo aggiornamento. Se nel nuovo istante temporale il valore del MSE è aumentato, il parametro μ viene incrementato, se invece il MSE cala, μ viene decrementato.

Sono stati valutati altri approcci, alternativi al *Gradient Descent* : in particolare è risultato molto performante l'algoritmo di *Levenberg-Marquardt*. L'algoritmo è il giusto compromesso fra complessità e velocità di convergenza. È una via di mezzo fra il *Gradient Descent* e il metodo *quasi-Newton* che si basa sulla valutazione della derivata seconda anziché la derivata prima. L'approccio *quasi-Newton* velocizza enormemente la convergenza, riducendo le iterazioni necessarie, ma introduce un problema computazionale non indifferente. *Levenberg-Marquardt* aggira il problema del calcolo di una derivata seconda riconducendola ad un'equivalente derivata prima, conservando il vantaggio della rapidità di convergenza. Inoltre presenta la miglior minimizzazione del MSE [10].

Iterando il procedimento visto per l'aggiornamento dei pesi, per ciascun esempio del *training set*, si completa l'algoritmo di *back-propagation*, con l'obiettivo di minimizzare sempre più la funzione MSE. Il processo di apprendimento del percettrone viene quindi condotto a cicli iterativi. La fase di addestramento può essere terminata quando l'errore commesso dalla rete scende sotto una determinata soglia oppure dopo un numero prefissato di iterazioni [11].

È infatti importante evitare il fenomeno dell'*overtraining,* in cui un'eccessiva fase di *training* porta ad un'esagerata focalizzazione sul *dataset* e una mancanza di generalizzazione. La conseguenza dell'*overtraining* è detta *overfitting.*

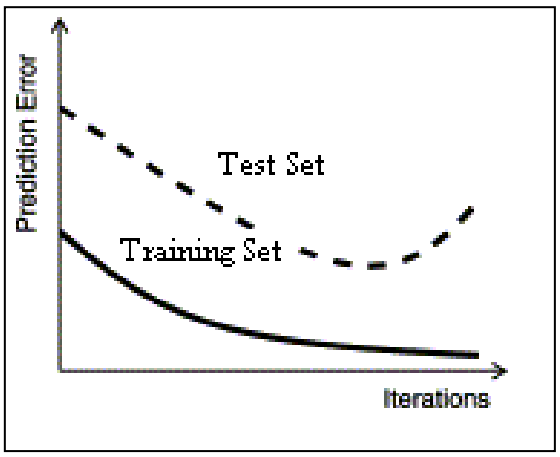

Figura 13: Esempio di overfitting.

Per queste ragioni si introduce una terza fase posta fra la *training* e il *test*, detta v*alidation Set*. I criteri di uscita dalla fase di *training* sono importanti parametri di progetto; ad esempio il numero di iterazioni, il tempo trascorso, il valore del gradiente, etc.

Le reti neurali sono *data-dependent*, cioè le prestazioni dipendono in maniera determinante dalla bontà dei dati di input, il *dataset*. Risulterà dunque fondamentale garantire un *dataset* il più possibile rappresentativo del funzionamento del sistema.

2.2 Reti neurali nella FDI

Come anticipato in precedenza, le reti neurali si distinguono per l'ottima capacità di adattamento al problema. Riescono infatti a trovare le relazioni nascoste che legano input e output. Fra le numerose applicazioni delle reti troviamo anche la diagnostica, in cui risulta fondamentale scoprire i legami fra i segnali in gioco nel sistema e i guasti che si possono verificare. Le reti neurali, già dagli anni '90, hanno trovato largo impiego nel settore, ispirando numerose ricerche in ambito scientifico.

Le applicazioni possono essere inquadrate in due categorie principali:

- *Pattern recognition,* o classificazione.
- *Prediction,* o predizione.

2.2.1 Pattern recognition

Il vettore **i** in ingresso, composto da *h* elementi, rappresenta i campioni provenienti da *h* sensori e viene detto *pattern*. La tecnica di *pattern recognition*, o classificazione, consiste nell'associazione di un *pattern* d'ingresso ad una o più classi in uscita, nel nostro caso modi guasto.

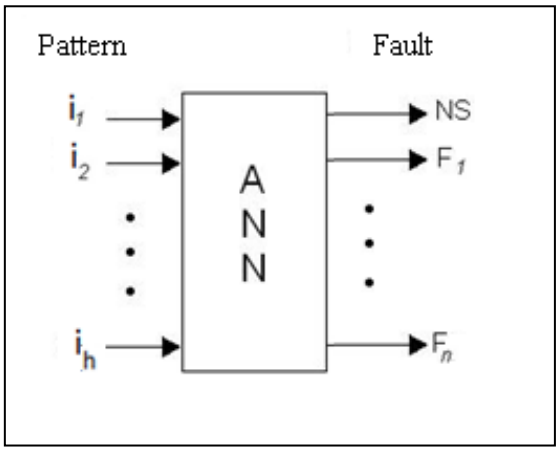

Figura 14: Pattern recognition.

I *pattern* che appartengono alla stessa classe sono accumunati da alcune proprietà o caratteristiche. Ad esempio possiamo individuare una classe composta da tutte quelle immagini che contengono la stessa persona, oppure la classe delle canzoni composte da un certo autore.

La classificazione avviene per mezzo di un classificatore che viene addestrato su un insieme di esempi, il *training set*. L'apprendimento può essere ricondotto a tre diverse approcci:

- apprendimento supervisionato:
  il *training set* è composto da esempi etichettati, cioè ad ogni esempio è già associata la classe di appartenenza.
- apprendimento non supervisionato:
  gli esempi del *training set* non sono auto-classificati. Si ricorre al problema statistico di stima di densità di probabilità.
- apprendimento con rinforzo:
  il classificatore apprende mentre viene utilizzato, osservando i risultati ottenuti. Per esempio, in un gioco, valutando i risultati delle mosse precedenti.

Risulta dunque fondamentale il concetto di *inferenza*, cioè quel processo attraverso il quale, da un esempio accolto come vero, si passa ad un secondo esempio la cui verità è dedotta dal contenuto del primo.

Il sistema è progettato in maniera tale da rilevare *fault* già avvenuti nella fase di addestramento. Di conseguenza *fault* sconosciuti alla rete verranno considerati come normali condizioni operative. Si intuisce l'importanza della fase di *training* e quindi del *dataset* che dovrà essere il più rappresentativo possibile del *range* di funzionamento della macchina [12].

L'approccio *pattern recognition,* seppur molto interessante, si scontra con requisiti spesso problematici. Infatti un *training set* auto-classificato è una risorsa di difficile reperimento. L'idea potrebbe essere quella di costruirselo con un parco macchine sufficientemente elevato.

Per una maggiore astrazione del problema e quindi rendere il sistema più flessibile è stato progettato un approccio a due stadi [13].

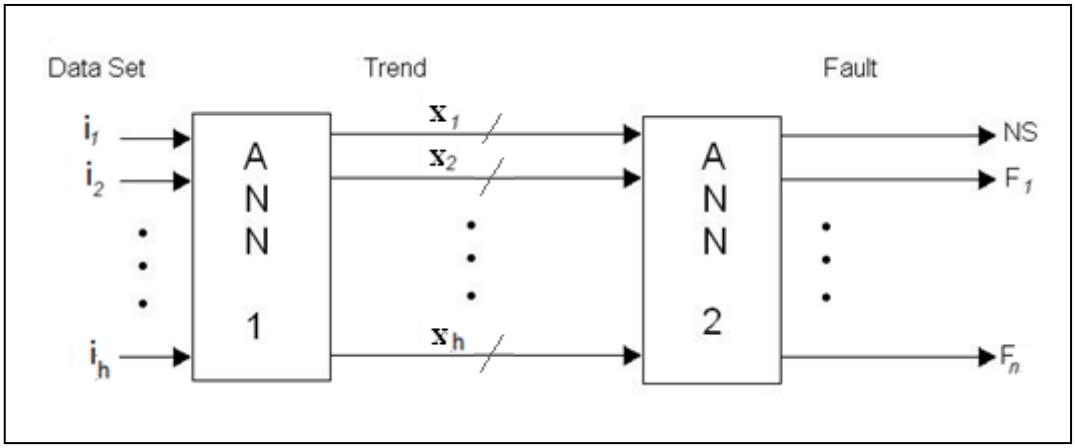

Figura 15: Pattern recognition a doppio stadio.

In ingresso alla prima rete neurale, ANN 1, è presente il *dataset* composto dagli *h* segnali di interesse. Il compito della prima rete è quella di riassumere il comportamento del singolo segnale all'interno di una finestra temporale, attraverso tre parametri. Tali parametri indicheranno l'entità della crescita, del calo o della costanza del segnale stesso, e saranno contenuti all'interno del vettore dei trend ***x***.

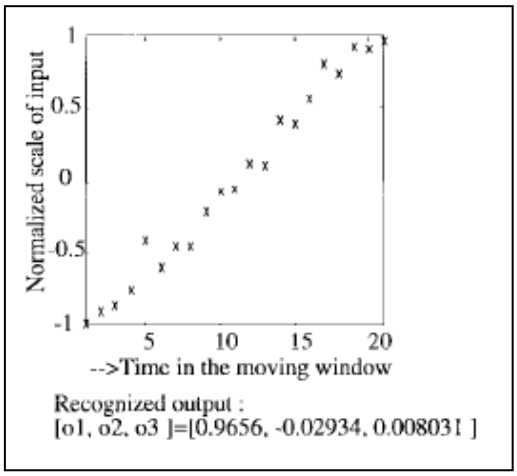

Figura 16: Esempio di tipico incremento.

La seconda rete, indipendentemente dalla prima, si occuperà dell'associazione fra i possibili trend dei diversi segnali con lo stato normale, NS, o i singoli *fault*.

2.2.2 One-step-ahead Prediction

Le reti neurali possono essere utilizzate per modellare sistemi MIMO, *Multiple Input Multiple Output*, di tipo non lineare. Dopo una prima fase di *training*, la rete, all'istante temporale *n*, è in grado di fornire la stima dell'uscita del sistema all'istante temporale *n+1*. In sostanza si utilizza il vettore $\boldsymbol{i(t_n)}$ che rappresenta il *dataset* dei sensori all'istante di tempo generico *n* e lo si invia ritardato di un istante all'ingresso della prima rete.

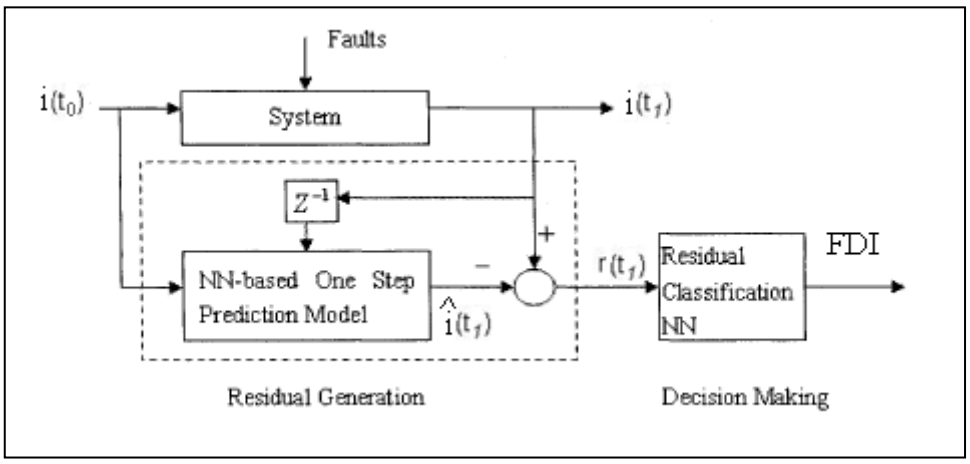

Figura 17: Prediction

. La rete stima il possibile vettore di uscita all'istante successivo, simulando il comportamento del sistema. La rete, previo addestramento su un *dataset* privo di malfunzionamenti, simulerà il comportamento nominale del sistema, a meno di inevitabili *range* di confidenza. Successivamente si effettua un confronto fra il vettore reale, proveniente dai sensori, e quello stimato, in uscita dalla rete, con la conseguente generazione di un parametro fondamentale: il residuo r.

Il residuo è un parametro che fornisce importanti informazioni sullo scostamento tra il comportamento attuale e quello nominale. Quando l'ampiezza del residuo supera una soglia predefinita si rileva il *fault*. Su questo parametro occorrerà quindi applicare la *teoria della decisione.*

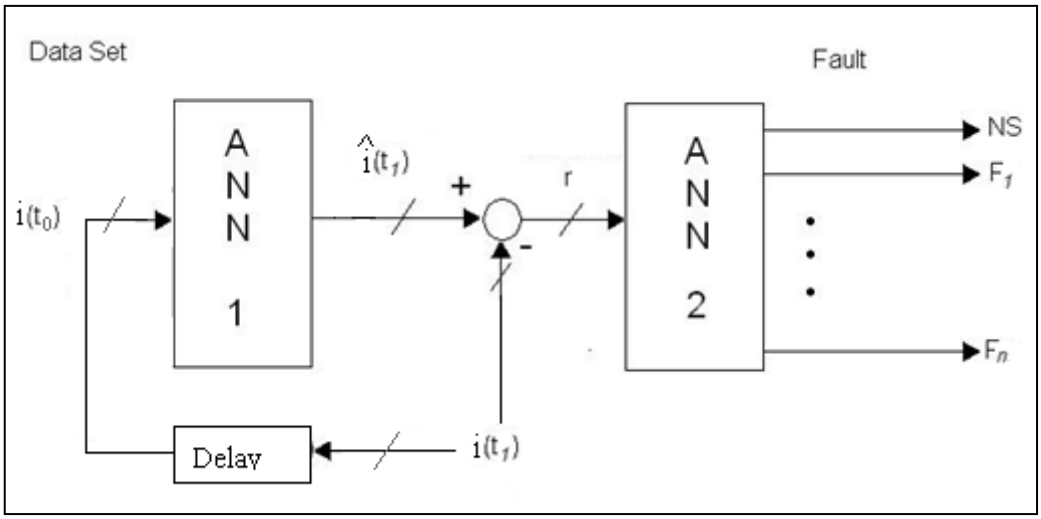

Figura 18: *Prediction* attraverso reti neurali.

Dal vettore dei residui ***r*** è possibile effettuare un'ultima elaborazione, in modo da implementare la funzione di isolamento dei *fault*. Concettualmente si tratta di un problema di classificazione. Questo compito viene assegnato alla seconda rete neurale. In realtà tale soluzione è stata semplificata preferendo alla rete neurale, di non semplice realizzazione, un approccio di tipo *fuzzy,* che verrà approfondito nel paragrafo 2.5.

È opportuno che il *dataset* in ingresso non contenga solamente *h* campioni riferiti al solo istante precedente degli *h* sensori. È più utile, infatti, riportare in ingresso più istanti temporali precedenti, in modo da rendere più efficienti le predizioni. Il numero di istanti temporali considerati dalla rete diventa un interessante parametro di progetto nella predizione.

La struttura denominata *one-step-ahead prediction*, o predizione un passo avanti, utilizza dunque uscite di istanti passati per predire l'uscita corrente [14].

2.3 Neural Network Toolbox ™

Matlab© mette a disposizione un pacchetto software molto utile per la progettazione, l'addestramento e la verifica delle reti neurali. Vengono messe a disposizione quattro possibili applicazioni delle reti neurali, ma due in particolare sono state utilizzate: *Time Series Tool e Pattern Recognition Tool*. Il primo *tool* viene utilizzato per implementare la predizione, il secondo per la classificazione [15].

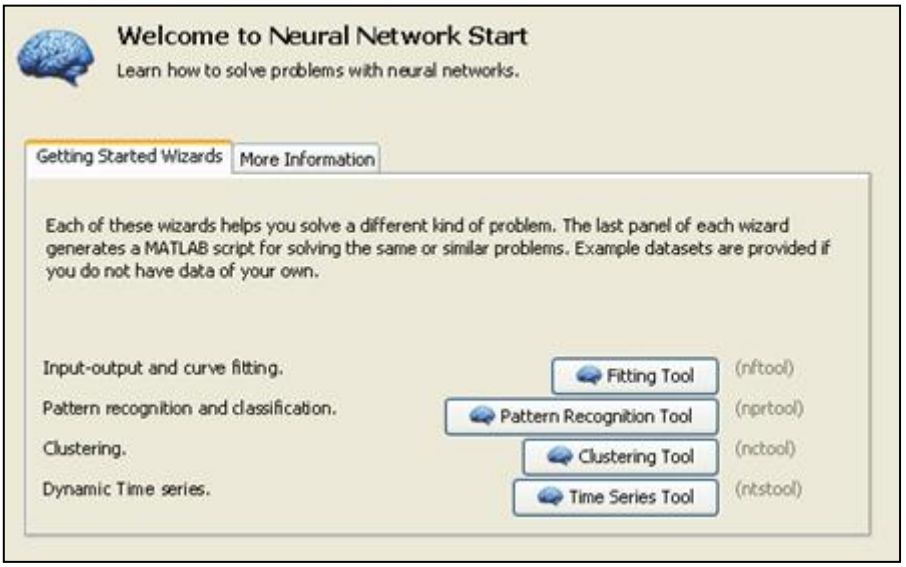

Figura 19: Neural Network Toolbox ™

Matlab© inoltre mette a disposizione alcuni esempi di *dataset* in modo da potersi esercitare sull'implementazione delle reti.

Per la predizione e la classificazione sono stati implementati due *script*, presentati in Appendice A e B, in modo da valutare le prestazioni e dunque le potenzialità di questa tecnica.

Due sono gli elementi principali di progetto nello *script* realizzato, in Appendice A, per implementare la predizione.

Il primo si tratta di un filtraggio temporale del *dataset*, necessario nel caso in cui si verifichi un ripple elevato.

Il secondo è rappresentato dai criteri di uscita della fase di apprendimento. Come spiegato nel paragrafo 2.1.4, è importante inserire limiti superiori alla fase di apprendimento per evitare il fenomeno di *overfitting*. Attraverso i seguenti parametri si impostano i criteri:

net.trainParam.epochs :	numero iterazioni massime, dette epoche.
net.trainParam.time :	tempo massimo.
net.trainParam.goal :	soglia minima del *Mean Square Error*.
net.trainParam.min_grad :	soglia minima del gradiente.
net.trainParam.mu_max :	massimo valore di μ.
net.trainParam.max_fail :	numero massimo fallimenti obiettivi in fase di *validation*.

Nell'approccio predittivo si evidenziano, in Figura 20, i dettagli della rete neurale implementata e l'evoluzione dei diversi parametri nella fase di *training*.

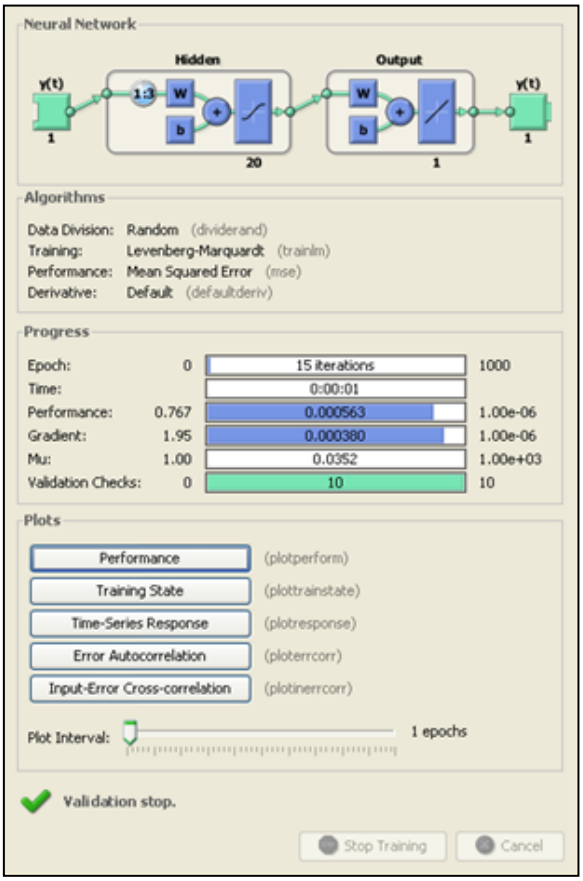

Figura 20: Parametri di uscita dall'addestramento.

In Figura 21 si è graficato l'andamento del MSE in funzione del numero di iterazioni, o epoche, effettuate. Si può osservare un netto calo nelle prime iterazioni e poi il raggiungimento di un *floor*.

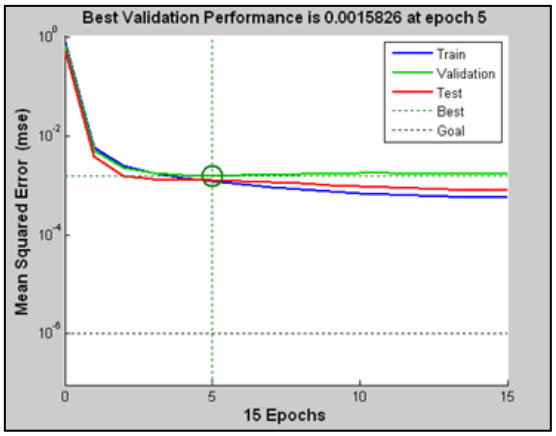

Figura 21: Mean Square Error per ciascuna iterazione.

Infine in Figura 22 si evidenzia, nel primo grafico, il confronto fra l'andamento temporale del segnale da monitorare e la previsione della rete. Nel secondo grafico invece si sottolinea l'errore commesso, cioè il residuo, ottenuto con la sottrazione fra i due andamenti precedenti. L'errore massimo commesso in questo caso è il 10% del *range* del segnale mentre il valore del RMSE nella fase di *test* è pari al 2,7%.

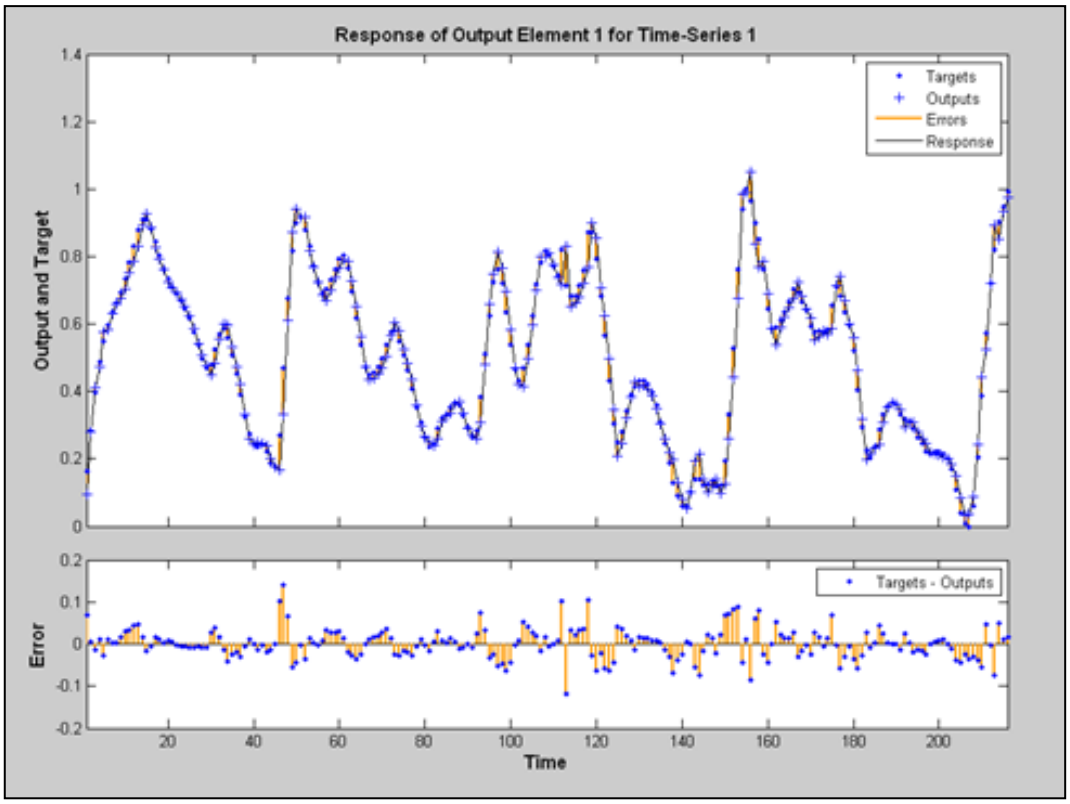

Figura 22: Andamento temporale del segnale predetto e dell'errore.

Per quanto riguarda la classificazione è stato implementato uno *script* per valutarne le prestazioni, presentato in Appendice B. Utilizzando come *dataset* uno degli esempi proposti da Matlab©, "glass_dataset", si è implementata una rete neurale capace di classificare in maniera corretta il 98.6% degli esempi:

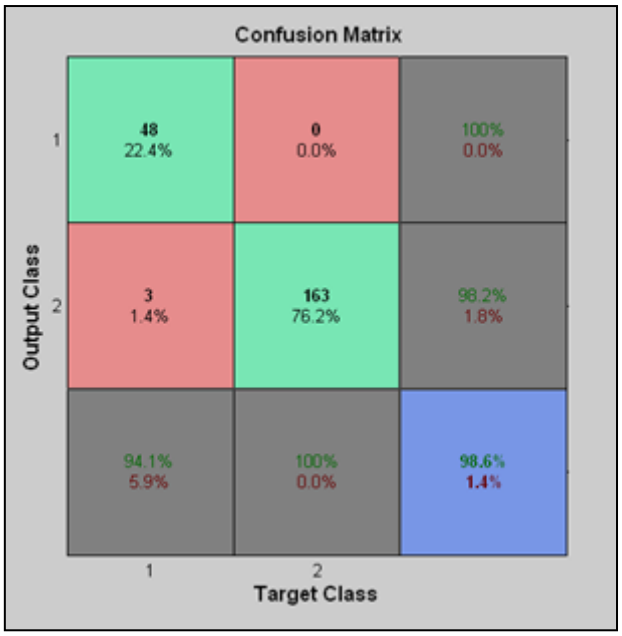

Figura 23: Risultati classificatore.

2.4 Implementazione algoritmo

Nella rete neurale si è individuato uno strumento molto utile per l'analisi e il controllo del profilo temporale di segnali di interesse.

In particolare si è voluto studiare, attraverso opportune simulazioni, il comportamento della rete in presenza di alcune anomalie; da un lato comuni, dall'altro facilmente simulabili. Nei seguenti esempi è stata progettata una rete neurale composta da 100 percettroni e 10 linee di ritardo.

In Figura 24 è riportato l'andamento temporale del segnale che è stato utilizzato per la fase di *training* della rete. È possibile osservare come l'errore massimo compiuto sia di 0.007 su un range di 6, quindi un errore molto ridotto dello 0.12%.

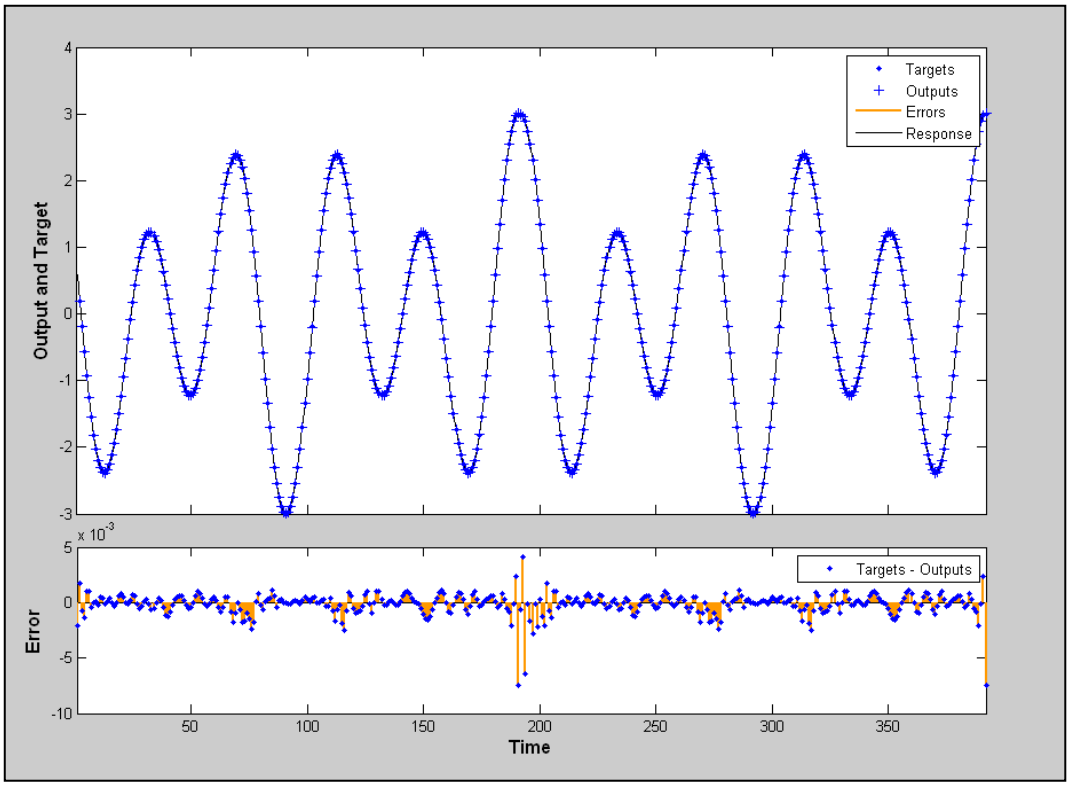

Figura 24: Fase di test in condizioni nominali.

Ora ipotizziamo che un generico *fault* introduca un *offset* all'interno del *dataset*. In Figura 25 si osserva la presenza dell'*offset* a partire dall'istante 400. L'errore risulta molto consistente nei primi 10 residui a causa della discontinuità. È importante notare come il 10, sia anche il numero di ritardi temporali della rete neurale. Questo chiaramente non si tratta di una coincidenza: infatti superati i 10 istanti temporali la rete non ha più memoria della discontinuità avvenuta in passato. In ogni modo anche successivamente si può osservare un errore consistente dell'ordine del 17%. L'anomalia simulata viene dunque correttamente rilevata dalla rete neurale progettata.

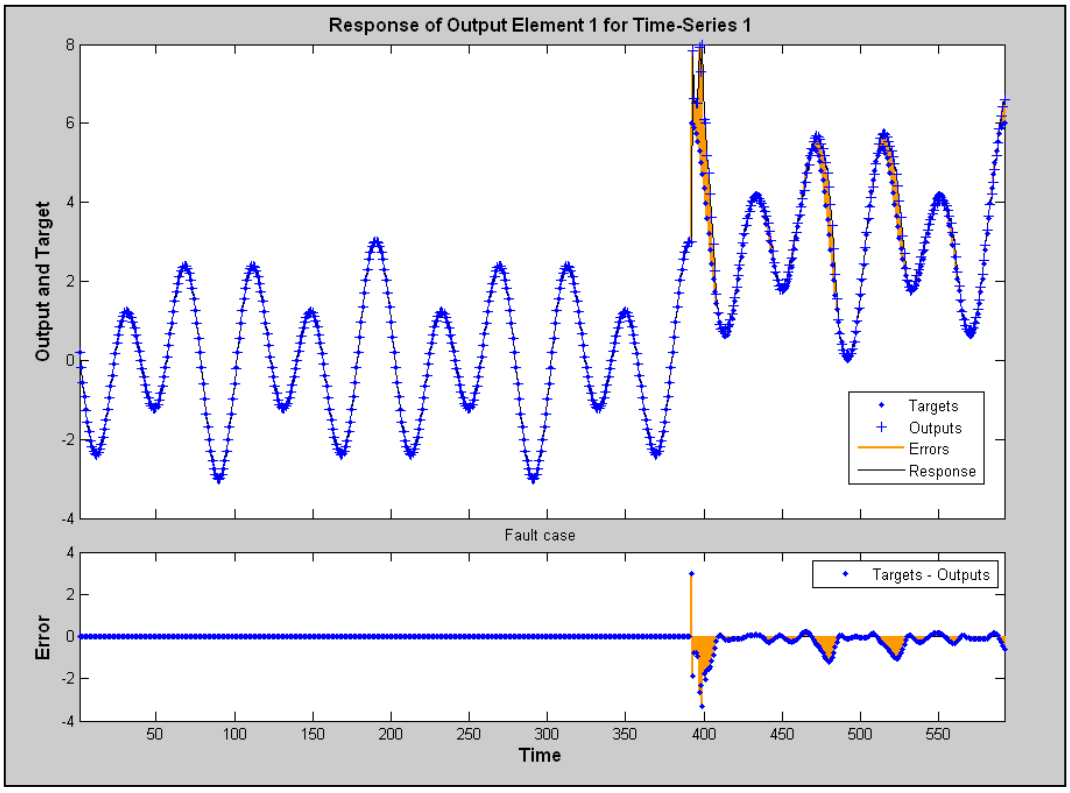

Figura 25: Fase di test in presenza di offset.

In Figura 26 si è simulato un cambio di periodicità del segnale, sempre a partire dall'istante 400. Anche in questo caso la rete si accorge in modo corretto dell'irregolarità segnalando un residuo del 18%.

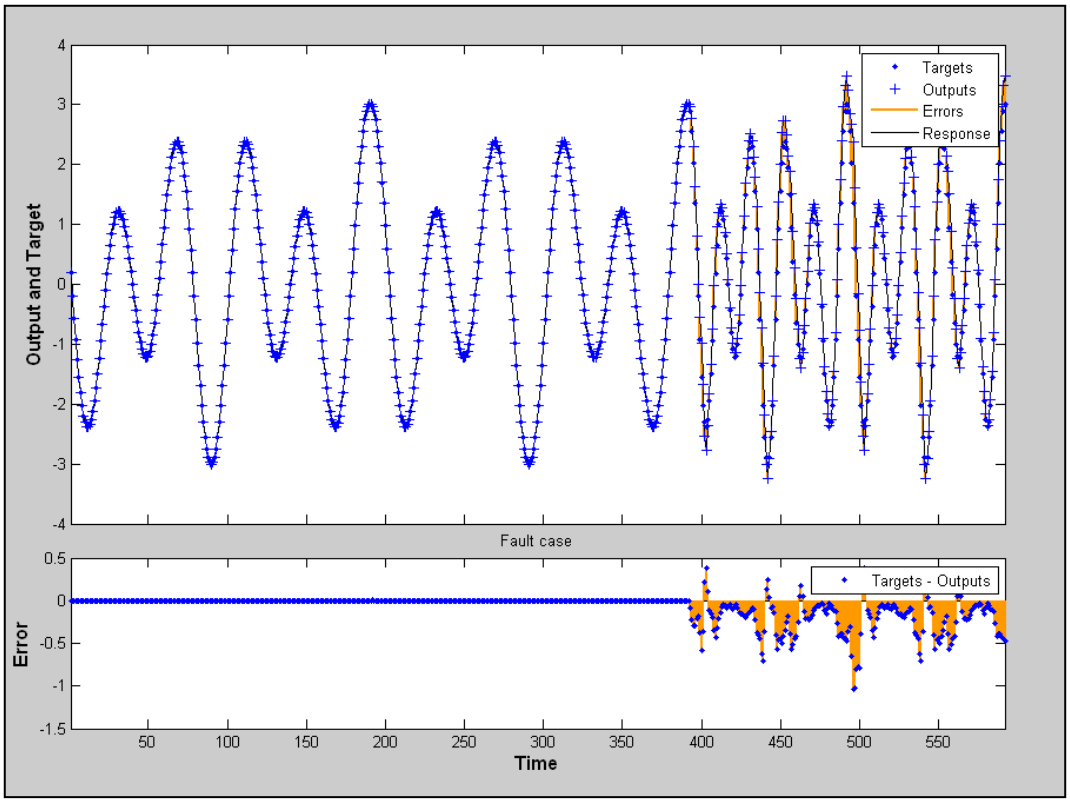

Figura 26: Fase di test in caso di modifica della periodicità.

Infine in Figura 27 si può osservare la modifica della forma d'onda, a causa di un cambiamento di una delle componenti spettrali. Anche in questo caso i 10 istanti intorno al 400° presentano residui più ampi a causa della discontinuità introdotta. L'anomalia, anche in questo caso, è stata correttamente segnalata dalla rete.

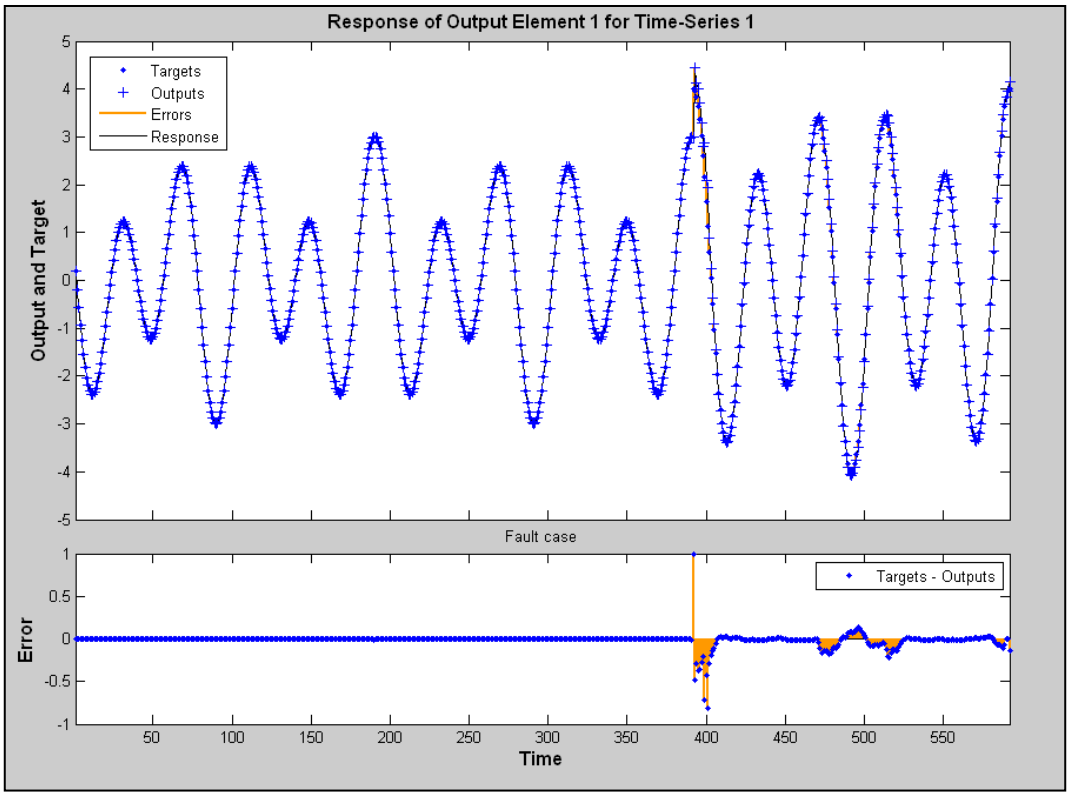

Figura 27: Fase di test in caso di modifica delle componenti spettrali.

In Figura 28 si è evidenziato lo zoom della Figura 27, per apprezzare l'ampiezza del residuo successivamente all'intervallo di memoria dei 10 istanti.

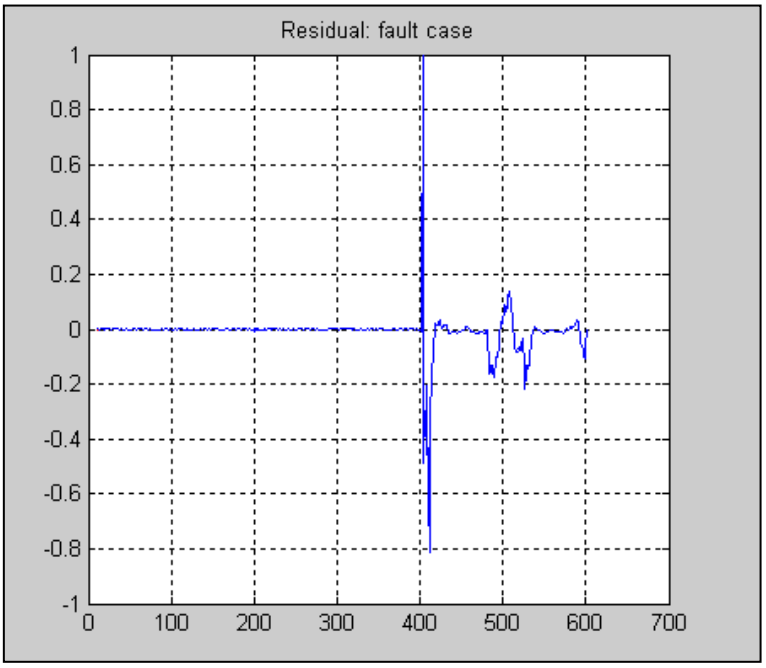

Figura 28: Zoom dei residui.

Dopo le fasi precedenti di simulazione e analisi, in cui si è utilizzata la preziosa *toolbox* di Matlab©, si è estrapolato l'algoritmo della rete neurale e lo si è implementato nel linguaggio di programmazione Java©. L'algoritmo che utilizza la rete neurale, in particolare la fase di *test*, rappresenta il cuore del programma predittivo. Invece la fase di *training* e di conseguenza la creazione della rete stessa, a causa dell'intrinseca complessità, rimane compito della *toolbox* di Matlab©. Il codice implementato per le due fasi è presentato in Appendice A.

2.5 Logica *Fuzzy*

La valutazione dei residui è fondamentalmente un problema di classificazione e rappresenta il punto cruciale della FDI. Il collo di bottiglia dei sistemi diagnostici si trova nella progettazione delle soglie poiché si tratta di un continuo compromesso fra una bassa probabilità di falsi allarmi e una buona sensibilità ai guasti [16]. I residui, pur contenendo informazioni importanti per la rilevazione di *fault*, sono contaminati da rumore e disturbi. Dovendo oltretutto mettere più residui in relazioni logiche tra loro, l'applicazione della logica booleana sembra quantomeno una forzatura. Appare dunque più opportuno l'utilizzo di una logica meno rigida in queste situazioni così vaghe ed imprecise.

Per questo motivo si è ritenuto conveniente l'introduzione della logica *fuzzy*, poiché tale logica mostra vantaggi proprio nella gestione di casi complicati, incerti e basati su informazioni incomplete. Attraverso la logica *fuzzy* sarà possibile inserire nei criteri di classificazione la conoscenza euristica [17].

Lo scopo della logica *fuzzy* è quello di associare lo spazio degli input ad uno spazio degli output attraverso dei costrutti logici *if-then*, dette regole. La caratteristica principale della logica in esame consiste nell'abilità di prendere decisioni basate su sfumature di grigio, anziché informazioni o bianche o nere. Infatti, al costrutto *if-then*, non si applicano operatori binari, ma elementi detti *fuzzy set*, che indicano cioè un grado di appartenenza ad una certa classe [18].

Ad esempio, alla temperatura dell'acqua possono essere associati tre valori che quantificano l'appartenenza alla classe "fredda","tiepida" e "calda", attraverso tre funzioni di appartenenza. In Figura 29 la linea grigia verticale indica la temperatura in esame. A tale temperatura si associa la terna (fredda:0.9, tiepida:0.1, calda:0.0).

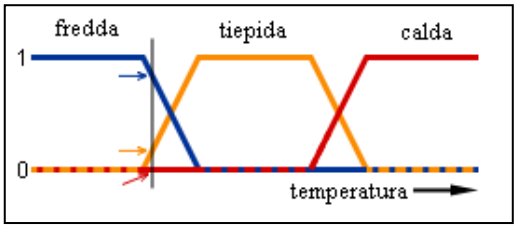

Figura 29: Funzioni di appartenenza.

Il mondo *booleano* e il mondo *fuzzy* presenteranno dunque delle equivalenze. La classica operazione logica "*and"* tra due variabili binarie corrisponde ad una funzione di *min()* tra due variabili reali comprese fra "0" e "1". Allo stesso modo un *"or"* logico coincide con la funzione di *max()*.

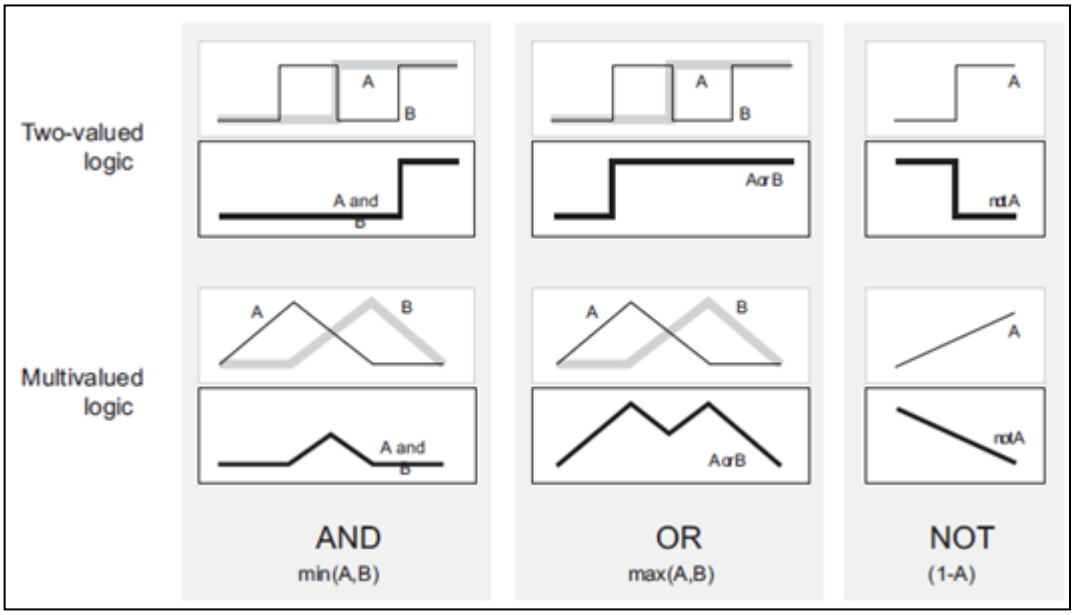

Figura 30: Operatori nella logica *booleana* (sopra) e *fuzzy* (sotto).

La logica *fuzzy* si divide in 3 fasi principali:

- *fuzzificazione*: agli ingressi vengono associati dei valori da "0" a "1" attraverso la funzione di appartenenza.
- applicazione regole: al *fuzzy set* si applicano gli operatori opportuni per le regole desiderate.
- *defuzzificazione*: si aggregano i diversi *fuzzy set* di output e si estrae il dato desiderato.

In Figura 31 si trova un esempio di una banale applicazione della logica *fuzzy*: a partire dalla qualità del cibo e del servizio si ricava la giusta mancia da lasciare al cameriere.

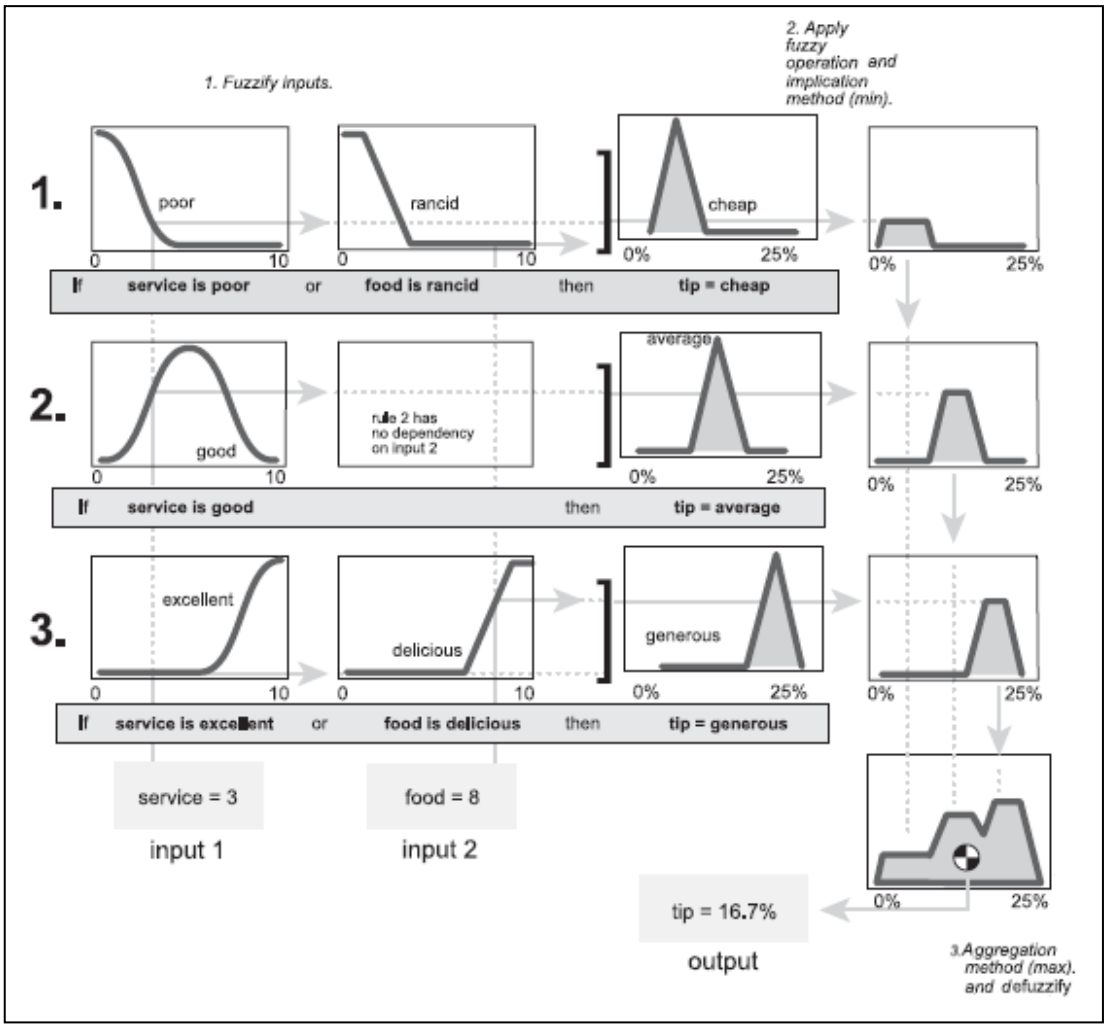

Figura 31: I 3 passi della logica fuzzy in un esempio pratico.

I due input, qualità del cibo e qualità del servizio, con valori compresi tra 0 e 10, vengono *fuzzificati* attraverso 3 funzioni in 3 diverse classi. In sostanza si quantifica, in un intervallo da 0 a 1, l'appartenenza del cibo o del servizio ad una determinata classe: scadente, buono o eccellente.

Nella seconda fase si applicano gli operatori *fuzzy* e l'implicazione dell'*if-then*. Nella logica *fuzzy* tale implicazione corrisponde ad una semplice funzione di *min()* tra il risultato del costrutto "*if*" e la funzione di appartenenza associata al costrutto "*then*".

Infine l'aggregazione, che corrisponde ad una *max()* dei diversi output, e la *defuzzificazione* attraverso il metodo del centroide (possono essere utilizzate altre tecniche).

L'idea dunque è quella di applicare il criterio *fuzzy*, appena visto, al vettore dei residui della rete neurale per la predizione. Matlab© consente la progettazione di un sistema basato su logica *fuzzy* attraverso una *toolbox* interamente dedicata. La logica viene progettata attraverso tre schermate di configurazione. Nella prima, evidenziata in Figura 32, si decidono il numero di ingressi, di uscite e i criteri di valutazione da adottare.

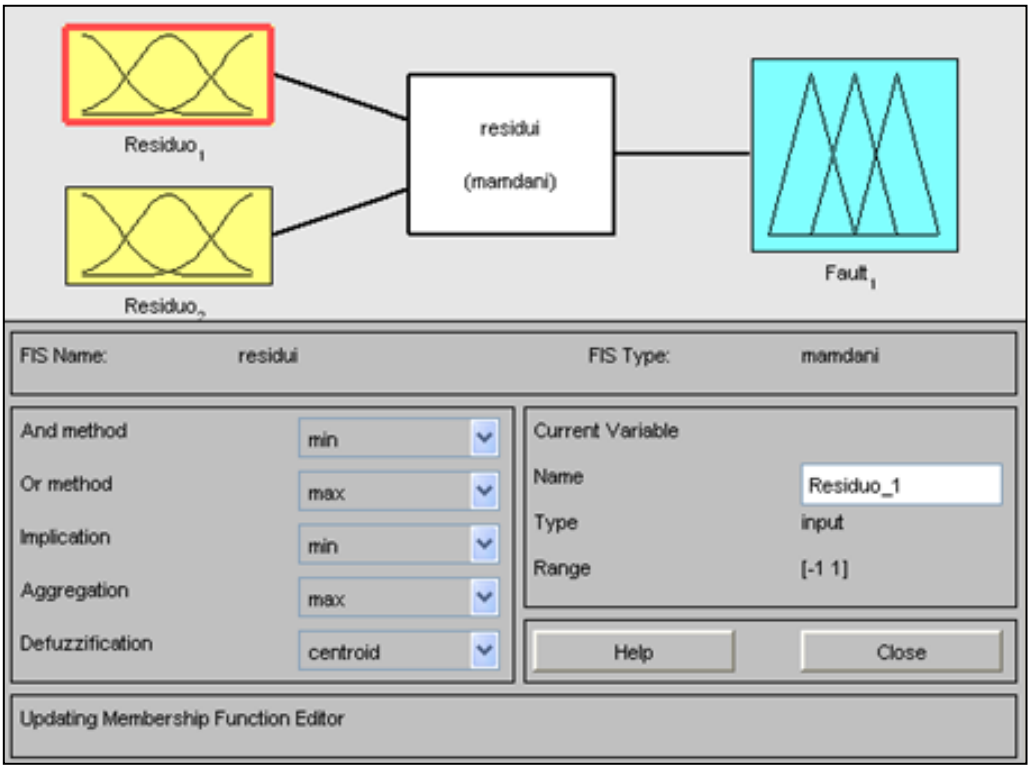

Figura 32: Configurazione sistema fuzzy.

Attraverso la seconda schermata, in Figura 33, è possibile progettare le diverse funzioni di appartenenza. Tali funzioni hanno un ruolo chiave nelle prestazioni della FDI. La forma e i parametri che le caratterizzano devono essere, infatti, opportunamente progettate per minimizzare le probabilità di *miss detection* e *false alarm.* In questo caso è stata scelta la funzione gaussiana.

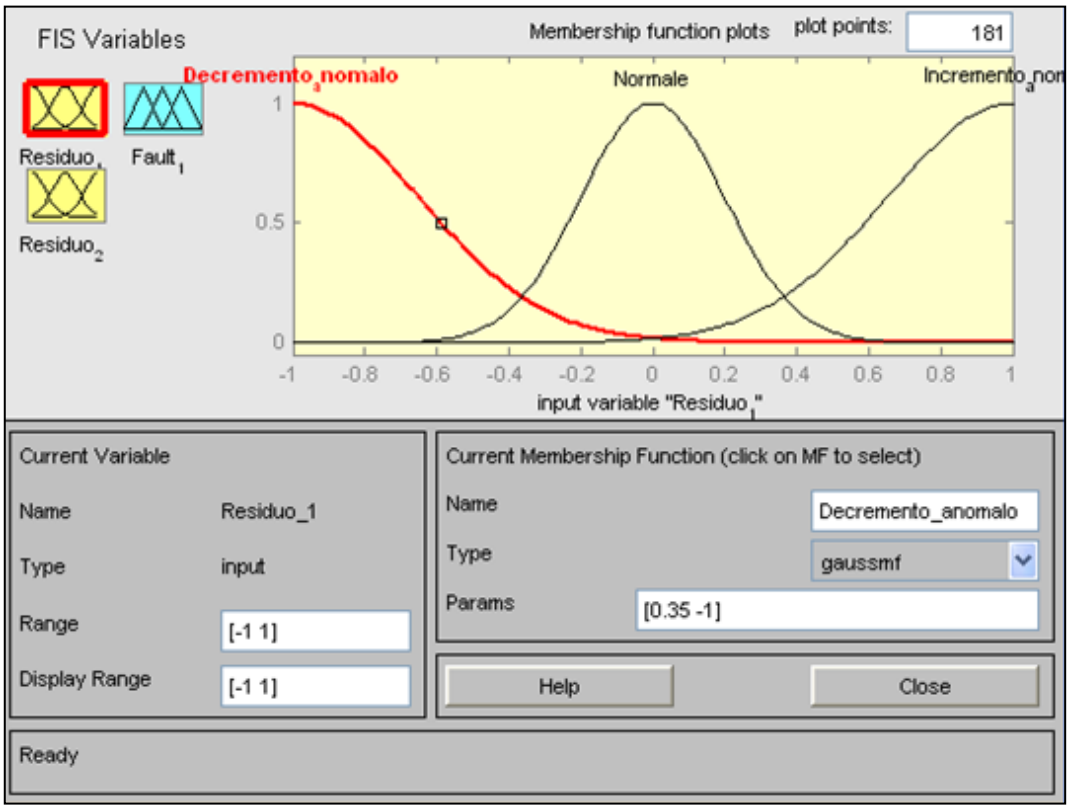

Figura 33: Progettazione funzioni di appartenenza.

L'ultima schermata consente l'introduzione delle regole che mettono in relazione i residui con i *fault*. Ipotizzando la presenza di un generico *fault* in corrispondenza di un contemporaneo incremento anomalo del residuo n°1 e decremento anomalo del residuo n°2, si inserisce la regola come in Figura 34.

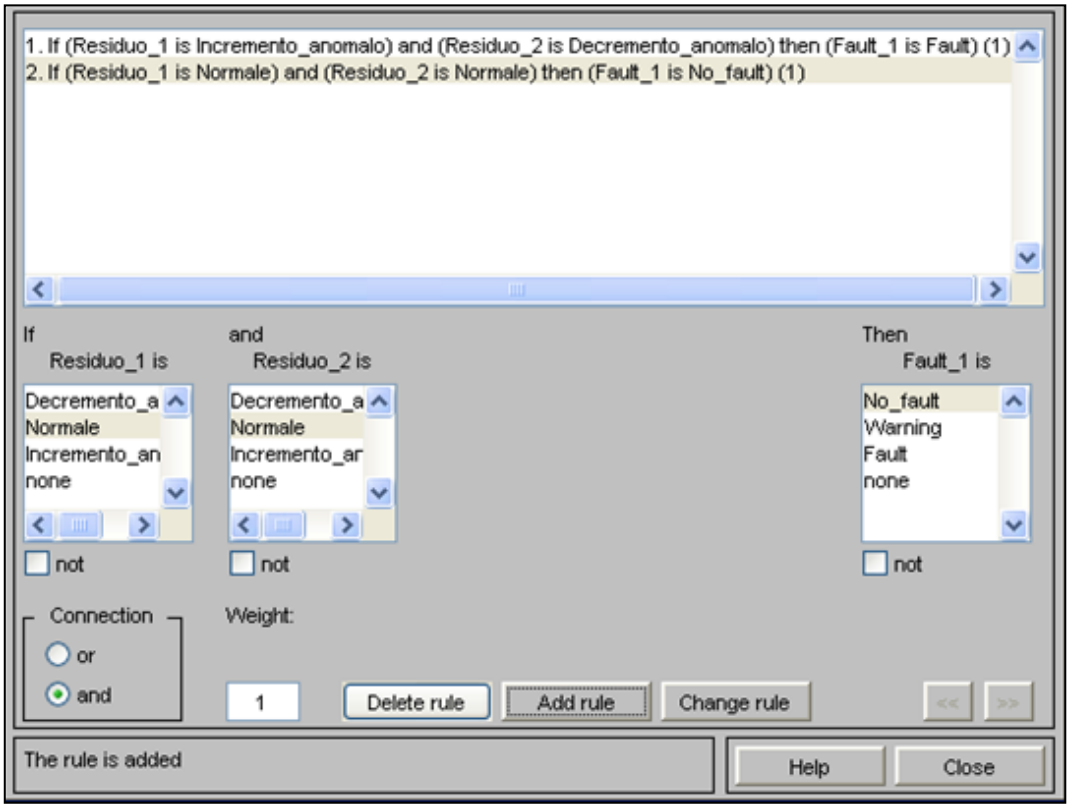

Figura 34: Inserimento regole.

Il costrutto *if-then*, quindi, non utilizza più la logica *booleana*; gli operandi sono rappresentati da un valore, compreso tra 0 e 1, che riflette l'appartenenza del residuo n°1 alla categoria "incremento anomalo".

In Figura 35 e Figura 36, rispettivamente, si rappresentano le due situazioni limite: condizioni operative nominali e *fault* rilevato. La linea rossa rappresenta il valore dei due residui mentre l'area gialla quantifica l'appartenenza alla relativa classe.

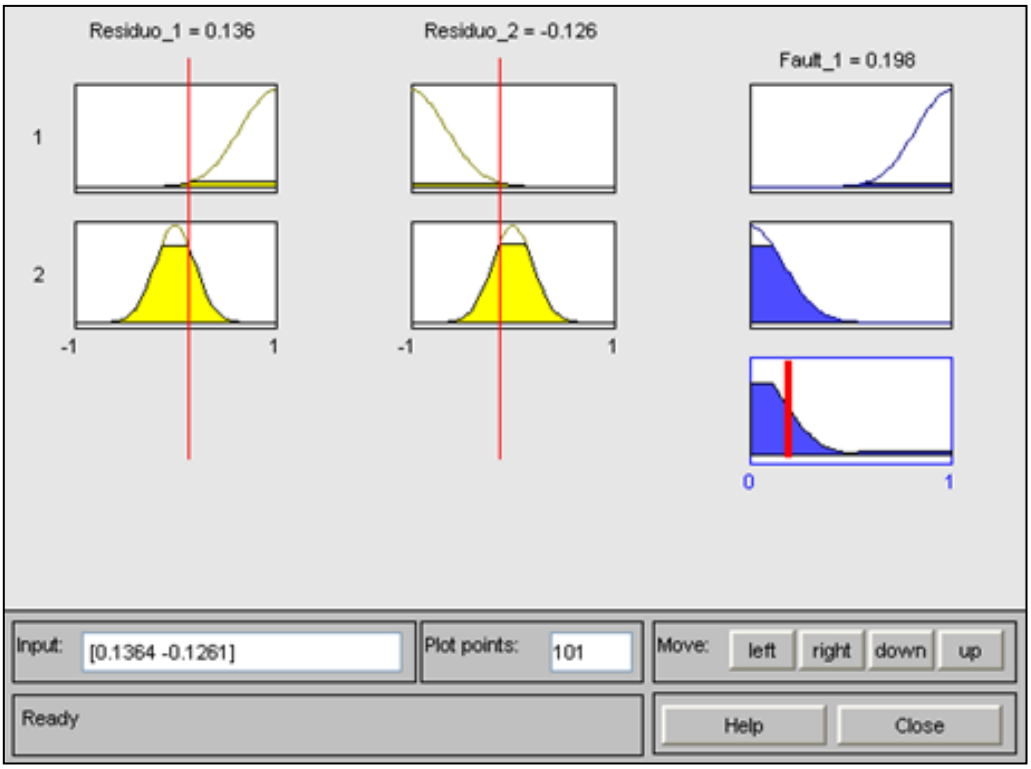

Figura 35: Esempio di condizioni operative nominali.

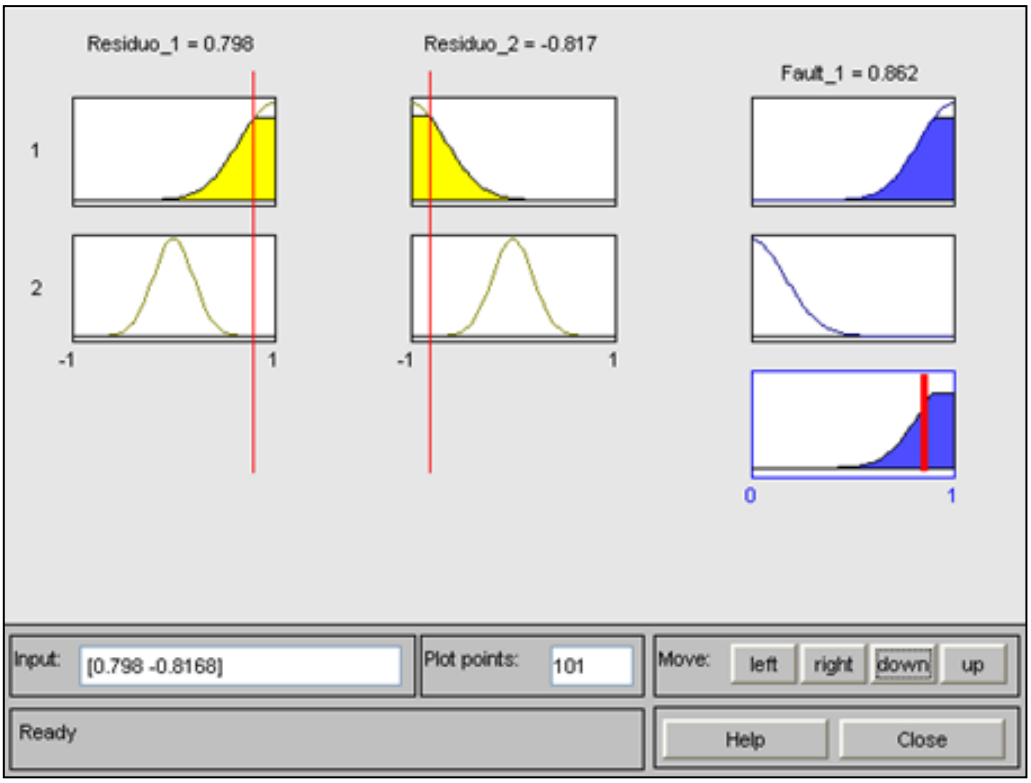

Figura 36: Esempio di *fault* rilevato.

Capitolo 3:

Approccio *signal-based*

Nell'attività di *Fault Detection and Isolation*, l'approccio *signal-based* prevede l'impiego della teoria dell'elaborazione dei segnali.

Il segnale può essere definito come una variazione temporale di una grandezza fisica. Il segnale quindi racchiude in sé una grande quantità di informazioni. La teoria dei segnali studia le loro possibili rappresentazioni in modo da analizzarne le proprietà matematiche e statistiche. Attraverso questa teoria è possibile ricavare le importanti informazioni che si nascondono negli andamenti temporali di alcune variabili del sistema. L'applicazione di questa teoria nell'attività di FDI consentirebbe quindi di automatizzare il rilevamento di possibili anomalie, altrimenti impossibile ad occhio nudo.

L'approccio *signal-based* cerca di scoprire quali variabili e quali relative informazioni possano essere associate ad eventuali guasti o malfunzionamenti. Si prendono in esame quindi quelle variabili che caratterizzano maggiormente il comportamento della macchina, come ad esempio le correnti di assorbimento, le velocità o le vibrazioni. Una volta in possesso del loro andamento temporale è possibile elaborare, attraverso opportuni algoritmi, tali valori. All'interno di una macchina automatica, l'andamento temporale di una variabile viene letto ad istanti discreti, subisce cioè un campionamento. Il segnale a disposizione si tratta di un segnale tempo-discreto, quindi l'elaborazione applicata in questo caso è di tipo numerico.

È opportuno fare una distinzione fra due dimensioni differenti applicare tale teoria:

- dominio temporale.
- dominio frequenziale.

3.1 Dominio temporale

Sono molteplici le possibili valutazioni da effettuare sui dati che descrivono l'andamento temporale di un parametro. In particolare si evidenziano i seguenti approcci:

- range:
  si individua un intervallo di valori entro il quale è ammissibile ricadano i campioni del parametro.

- variazioni:
  si prende in esame la differenza tra i valori di due campioni successivi.

- media:
  si calcola la media statistica del parametro in esame.

- varianza:
  si calcola la varianza del parametro, cioè il grado di dispersione attorno al suo valor medio.

È opportuno introdurre alcuni concetti poiché in queste valutazioni assume fondamentale importanza la *teoria della stima*.

Il problema della stima di un parametro si verifica ogni volta che si desidera effettuare un'inferenza, cioè trarre una conclusione, su un parametro a, dato il parametro osservabile r:

$$r = a + n \tag{19.}$$

dove *n* rappresenta una variabile aleatoria con distribuzione gaussiana.

Lo stimatore è dunque una funzione in grado di fornire delle stime che si avvicinano ai valori del parametro *a* da stimare.

Per valutare l'efficienza di uno stimatore si utilizza il *Mean Square Error*, MSE. La funzione MSE rappresenta un indice di performance della bontà di una stima. Per quantificare meglio la qualità dell'attività predittiva si preferisce la sua radice quadrata: il *Root Mean Square Error*, σ:

$$\sigma = \sqrt{\frac{1}{p}\sum_{h=1}^{p}(\hat{x}_h - x_h)^2} \qquad 20.$$

dove $\hat{x}$ rappresenta la generica stima e x la variabile osservabile.

È importante inoltre introdurre il concetto di intervallo di confidenza. Tale intervallo identifica un *range*, centrato sulla media campionaria, all'interno del quale ci si aspetta di trovare il parametro incognito. A ciascun intervallo di confidenza viene associato un livello di confidenza che indica il grado di attendibilità dell'intervallo stesso [19].

Assumendo, come ipotesi, una distribuzione normale dei dati, cioè gaussiana, all'intervallo di confidenza $\pm 1.96 \cdot \sigma$ è associato un livello di confidenza del 95%.

$$P(\hat{x} - 1.96 \cdot \sigma < x < \hat{x} + 1.96 \cdot \sigma) = 95\% \qquad 21.$$

dove P(x) è la densità di probabilità, x l'oggetto della stima e σ il RMSE.

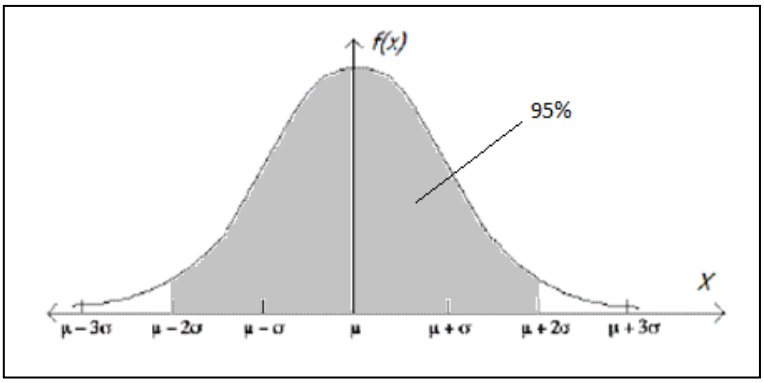

Figura 37: Distribuzione normale o gaussiana.

Quindi, per fare un esempio, nel caso in cui la stima del parametro sia $\hat{x} = 10$ e il *Root Mean Square Error* σ =1%, l'intervallo di confidenza associato ad un livello di confidenza del 95% è pari ad un intervallo di 0.0392:

$$P(9.9804 < x < 10.0196) = 95\% \qquad 22.$$

3.2 Dominio frequenziale

Per quanto riguarda la dimensione frequenziale vi è un passo intermedio da inserire fra la lettura dei dati ed una loro elaborazione, ossia la valutazione spettrale del segnale. È opportuno sottolineare come la valutazione dello spettro sia essa stessa un'elaborazione del segnale.

Oltre allo spettro della corrente del motore, analizzato fin dai primi test, si pensa di estendere in futuro tale valutazione, alle vibrazioni presenti nei diversi gruppi della macchina. La semplice aggiunta di alcuni accelerometri, a prezzi molto contenuti, consentirebbe la raccolta di preziose informazioni che si celano nelle vibrazioni presenti. I guasti di tipo meccanico, infatti, hanno quasi sempre ripercussioni a livello vibrazionale. L'analisi spettrale è uno strumento molto utilizzato negli ambiti diagnostici poiché consente l'individuazione di armoniche indesiderate, introdotte da eventuali anomalie [20].

Si è realizzato uno strumento di simulazione, sempre attraverso Matlab© (il codice in Appendice C), in grado di elaborare la densità spettrale di potenza del segnale in ingresso e costruire il residuo, frutto del confronto fra il comportamento spettrale nominale e quello attuale [21] [22]. Attraverso una successiva elaborazione dei residui, come ad esempio l'applicazione della logica fuzzy, vista nel paragrafo 2.5, è possibile completare l'attività di *detection* del *fault*.

Inoltre nel caso in cui si sia in grado di associare ad ogni *fault*, la frequenza in cui compare l'armonica indesiderata, è possibile, con un semplice filtro, effettuare anche l'attività di *isolation* del guasto.

L'algoritmo che sfrutta l'analisi spettrale è stato infine implementato nel linguaggio Java, in Appendice C.

In particolare, si ipotizza un segnale con componenti frequenziali a 60 e 200 Hz. Si può osservare in Figura 38, in alto, il confronto fra il caso nominale, in rosso, e quello attuale, in blu; i due spettri coincidono, a meno di piccolissime variazioni dovute al rumore. In maniera corretta, il residuo, in basso in Figura 38, non segnala alcuna anomalia.

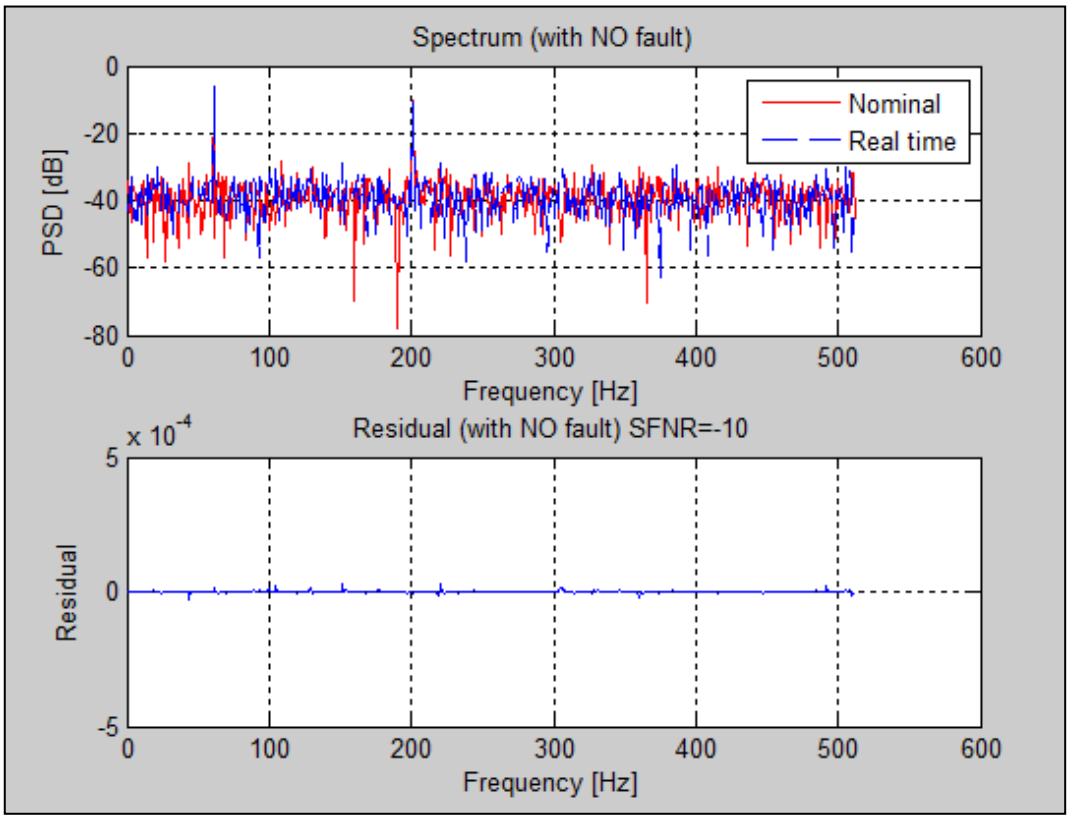

Figura 38: Analisi spettrale nel caso nominale.

In Figura 39, invece, si ipotizza la presenza di un *fault* che introduce un'armonica indesiderata a 110 Hz. Il residuo segnala correttamente la presenza di un'irregolarità proprio in corrispondenza dei 110 Hz. Il *fault* viene dunque rilevato prontamente. Se inoltre si fosse in grado di associare al *fault* simulato, la frequenza in cui compare l'armonica indesiderata, cioè 110 Hz, sarebbe possibile effettuare anche l'attività di isolamento del guasto.

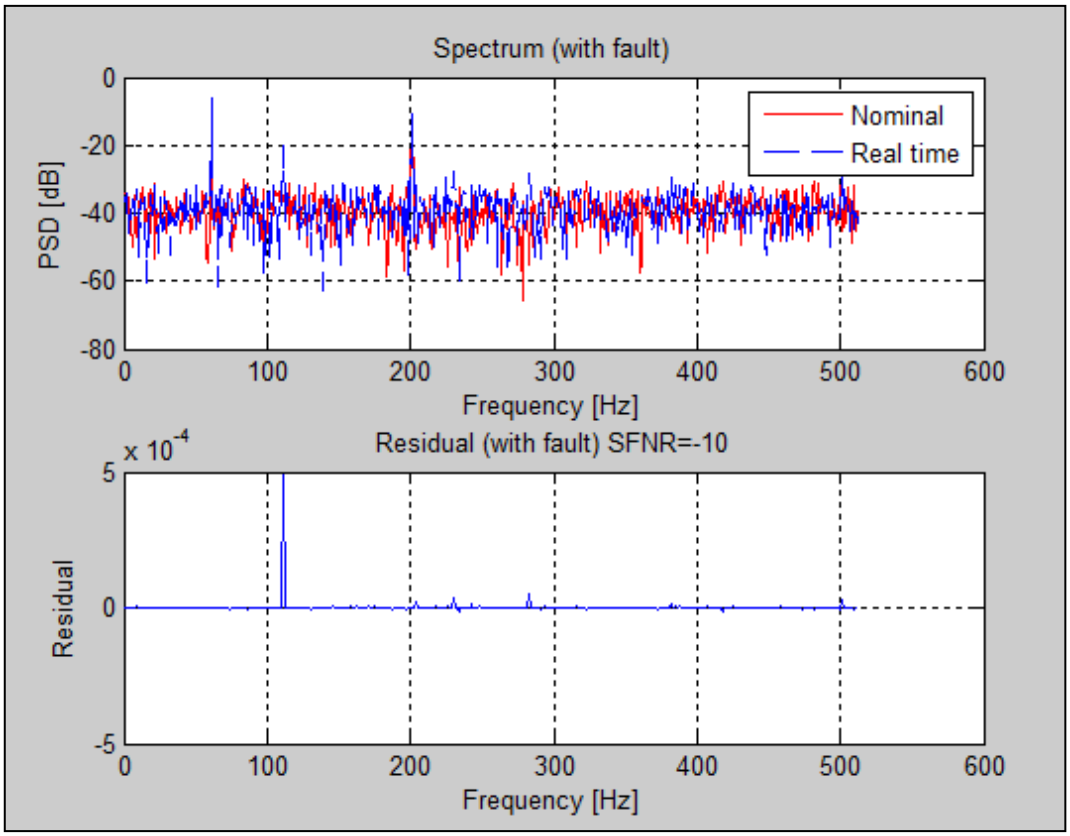

Figura 39: Analisi spettrale in caso di *fault*.

Capitolo 4:

La macchina automatica

Con il termine "macchina automatica" si intende un'entità autonoma, da un punto di vista funzionale, capace di svolgere un previsto insieme di funzioni, su un flusso di unità di prodotto che la attraversano. L'autonomia funzionale rappresenta la capacità di svolgere l'insieme delle funzioni previste in maniera indipendente dall'attività delle macchine a monte o a valle, purché sia garantita una corretta alimentazione della macchina [23]. È possibile scomporla in tre sottosistemi:

- sistema di rilevamento delle informazioni:
  una serie di sensori distribuiti sul campo genera un rilevante flusso informativo che ha lo scopo di aggiornare l'unità di controllo sull'evoluzione dello stato della macchina.
- sistema di controllo:
  a partire dal flusso in informazioni genera un adeguato flusso di comandi in modo da gestire opportunamente l'unità di attuazione.
- sistema di attuazione:
  sede del flusso energetico, grazie ai comandi ricevuti deve generare tutti i movimenti richiesti.

All'interno di un complesso sistema, il numero dei dispositivi che devono interagire tra di loro può essere molto elevato: *Programmable logic controller*, o PLC, sensori, attuatori, controllori *embedded*…

Risulta quindi facile immaginare come l'enorme mole di dati da scambiare possa rappresentare un "collo di bottiglia" per l'intero sistema. Inoltre a differenza di altri sistemi, quello di controllo, presenta uno stringente vincolo *real-time*, per cui l'avvenuto scambio di informazioni può risultare totalmente inutile se al di fuori della *deadline* prefissata. Le reti digitali tradizionali, progettate per reti locali e

globali di computer, non hanno quindi caratteristiche idonee per poter essere impiegate fino al livello di campo dell'impianto, cioè nelle comunicazioni tra sensori, attuatori e sistemi di controllo.

In realtà è possibile suddividere in maniera gerarchica la rete in 3 diverse livelli, come rappresentato in Figura 40:

- livello di supervisione:

  le *workstation* di supervisione sono collegate con il sistema informativo dei reparti dirigenziali attraverso reti informatiche aziendali. Le tecnologie sono quelle tipiche delle reti LAN, *Local Area Networks*.

- livello di controllo:

  le reti per il controllo svolgono il compito vitale di sincronizzazione tra i controllori locali di una cella produttiva e celle differenti. Per garantire la corretta sequenza degli eventi, deve essere garantito in maniera deterministica il ritardo di trasmissione. Normalmente tali reti sono basate su tecnologie di tipo proprietario per connettere controllori della stessa casa costruttrice.

- livello di campo:

  I dispositivi di controllo sono collegate a sensori e attuatori attraverso le reti di campo o *fieldbus*.

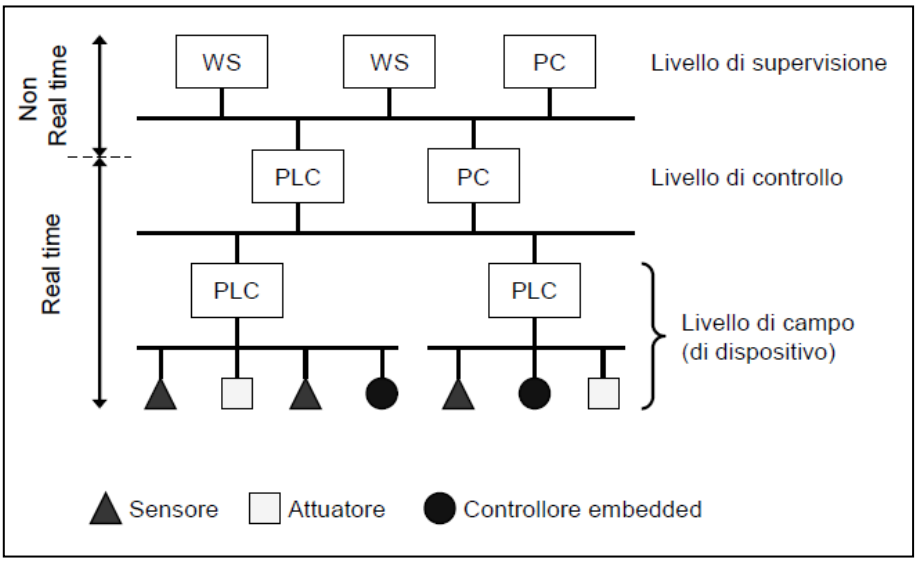

Figura 40: Classificazione delle reti per il controllo.

La suddivisione gerarchica è molto utile perché consente la progettazione specifica di ciascun livello, in base alle singole esigenze. Infatti, come evidente in Figura 41 sono molto diversi i compiti che devono assolvere.

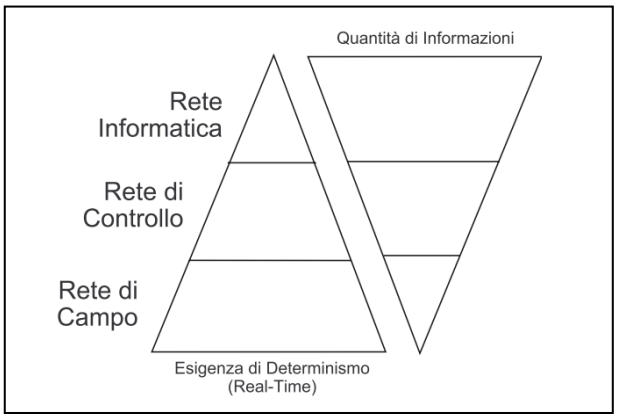

Figura 41: Confronto fra le esigenze delle diverse reti.

Per questo motivo nascono tipologie di rete e protocolli di trasmissione dati progettate appositamente per le comunicazioni orientate al controllo: le reti di campo o *fieldbus* [24]. Il nome non indica la topologia di connessione a bus, anche se normalmente viene utilizzata proprio quella. L'automazione industriale è l'applicazione principale ma sono utilizzate anche nelle applicazioni domotiche e del telecontrollo. Esiste una moltitudine di enti normativi che definiscono standard per i bus di campo; quasi tutte comunque sono confluite nella normativa IEC 61158, nata nella metà degli anni '80 per contenere la proliferazione di standard proprietari. Si è però verificato che alcune applicazioni standard non siano compatibili tra di loro.

I bus che rientrano nella IEC 61158 sono:

- *ControlNet*
- *Profibus*
- *P-NET*
- *Fieldbus Foundation*
- *WorldFIP*
- *SwiftNet*
- *Interbus*

4.1 Sistema di controllo

Il sistema di controllo rappresenta il cervello della macchina automatica. L'elaborazione del flusso di comandi deriva da due flussi informativi in ingresso. Innanzitutto al momento della progettazione, prima della messa in servizio della macchina, occorre immettere nel sistema di controllo la descrizione del comportamento richiesto alla macchina, attraverso opportuni linguaggi, introdotti più avanti in questo paragrafo.

L'altro flusso informativo riguarda il funzionamento della macchina: i sensori, opportunamente inseriti nel sistema, documentano l'evoluzione dello stato della macchina stessa [25].

Una rilevante funzione svolta dal sistema di controllo, oltre all'elaborazione dei comandi, è quella di interfacciamento con il restante sistema di produzione e, ancor più importante, con l'operatore umano attraverso il pannello HMI, *Human Machine Interface*.

Il PLC, acronimo di "controllore logico programmabile", è un sistema elettronico destinato all'ambito industriale, che consente l'interazione e la gestione di macchine automatiche. Generalmente è costituito dai seguenti componenti:

- modulo alimentatore: dispositivo che fornisce la corretta alimentazione al sistema.
- moduli di input/output: mezzi attraverso i quali il PLC rileva dati dai sensori, comanda le azioni agli attuatori e comunica con altri computer o PLC. Tutto ciò avviene tramite un bus di comunicazione secondo determinati protocolli.
- modulo processore: la CPU rappresenta il cervello del PLC ed è basata su una logica programmabile.
- terminale di programmazione: il PLC non prevede tastiere e schermi, quindi la sua programmazione avviene tramite dispositivi esterni con software forniti dal costruttore del PLC stesso.

Il PLC, prima di tutto, legge tutti gli ingressi dai moduli di input, poi elabora in sequenza le istruzioni di comando ed infine riporta i risultati ai moduli di output.

Si tratta di un sistema *general purpose* e *real-time*; caratteristiche fondamentali in un contesto in cui riconfigurabilità e risposta del sistema entro tempi predefiniti, rappresentano vincoli irrinunciabili.

La nascita di questo sistema avviene alla fine degli anni ’60, quando la *General Motors* emise una direttiva ai sui fornitori per uniformare la gestione delle sue catene di montaggio, in modo da velocizzare i cambi di produzione, la revisione e le riparazioni.

Prima di questa direttiva, infatti, tutto era gestito attraverso la logica cablata e quindi sistemi costituiti da componenti discreti (pulsanti, relè, contatti, sensori, ...), connessi tra loro tramite collegamenti fisici che determinavano quindi un cablaggio fisso. Le diverse ditte costruttrici proposero, per ciascun PLC, una grande varietà di linguaggi di programmazione alternativi, con approcci spesso completamente differenti. Il PLC ebbe grande successo commerciale e il numero di dispositivi crebbe enormemente nel giro di vent’anni. Cominciarono quindi a sorgere numerosi problemi di interoperabilità e si sentì sempre più forte la necessità di un processo di standardizzazione. Solamente nel 1993 venne introdotta una normativa, la IEC 1131-3, che definì per la prima volta uno standard per il controllo logico [26].

La normativa IEC 1131-3 prende in considerazione, non solo i linguaggi di programmazione, ma anche le specifiche dei dispositivi e i protocolli per la comunicazione. Le sue linee guida tendono a favorire la portabilità del software, introducendo concetti di programmazione strutturata e modularità, in modo da facilitare la verifica e il riuso del codice e, di conseguenza, la riduzione di costi e tempi di sviluppo.

Vengono definiti 5 linguaggi di cui tre grafici:

- LD: *Ladder Diagram*, o Linguaggio a contatti

  il linguaggio più utilizzato fino a pochi anni fa, poiché era la trasposizione informatica dei circuiti elettrici della logica cablata. Si utilizzano semplicemente i simboli logici disegnando gli schemi elettrici nel software di programmazione, senza più cablare i relè.

- SFC: *Sequential Functional Chart*, o Diagramma funzionale sequenziale

  consente una facile implementazione di una macchina a stati finiti.

- FBD: *Function Block Diagram,* o Diagramma a blocchi funzionali

  per l'implementazione di diagrammi circuitali.

e due testuali:

- IL: *Instruction List*, o Lista di istruzioni

  linguaggio di basso livello, molto simile all'*Assembler*.

- ST: *Structured Text*, o Testo strutturato

  linguaggio di alto livello, simile al *Pascal*.

Figura 42: PLC Elau

4.2 Sistema di attuazione

Il sistema di attuazione può essere schematizzato come una linea di flusso di energia di vario genere (elettrica, pneumatica, chimica, …) , interfacciato in una o più sezioni con i comandi, provenienti dall'unità di controllo, che hanno il compito di modificarne i parametri. In ultimo, attraverso l'azionamento avviene la trasformazione tra energia primaria ed energia meccanica. Parte essenziale dell'azionamento è l'attuatore vero e proprio: il motore. Se la generazione del moto avviene secondo leggi di moto fissate il sistema si dice "in catena chiusa", in Figura 43, altrimenti viene detto "in catena aperta".

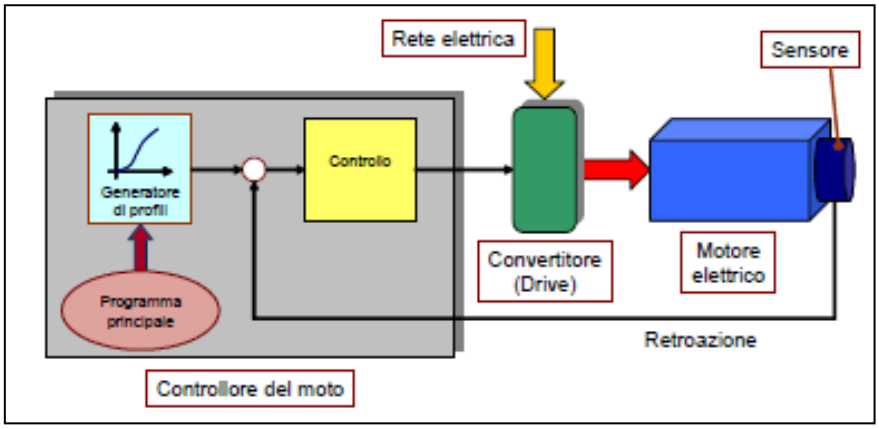

Figura 43: Schema di un azionamento in catena chiusa.

Il motore di tipo *brushless* ha grandissimo impiego nell'automazione industriale; risulta, infatti, particolarmente adatto nel caso di richieste di leggi di moto ad elevata dinamica ed elevata qualità nell'inseguimento di profili di moto prefissati.

Il moto, nel caso *brushless*, viene generato attraverso l'interazione elettromagnetica tra un rotore a magneti permanenti ed uno statore con degli avvolgimenti, i quali vengono eccitati in successione in maniera opportuna. Il nome deriva dalla novità di funzionamento introdotta, rispetto al motore a spazzole; non si basa più, infatti, su contatti elettrici striscianti sull'albero. La commutazione della corrente che circola negli avvolgimenti avviene elettronicamente e non più per via meccanica. In questo modo si riduce la resistenza meccanica e si elimina la

formazione di scintille all'aumento della velocità di rotazione, riducendo la periodica manutenzione.

Il motore *brushless* è di tipo sincrono poiché l'eccitazione dei singoli avvolgimenti dello statore avviene in modo sincrono con la posizione angolare del campo magnetico del rotore. Per rilevare la posizione angolare del rotore, e quindi tendere al sincronismo, si sfrutta un sensore di posizione angolare solidale al rotore stesso, come ad esempio il sensore ad effetto di Hall [27].

4.2.1 La corrente di assorbimento

Nella fase di sperimentazione si è posta l'attenzione principalmente sulla corrente di assorbimento del motore. La corrente di assorbimento è strettamente legata alla coppia motrice secondo la seguente formula:

$$I = \frac{C_m}{\eta\, K_t} \qquad 23.$$

dove C_m indica la coppia motrice, I la corrente di assorbimento, η il rendimento meccanico e K_t la costante di coppia del motore.

Vi è dunque un rapporto di proporzionalità diretta tra la corrente di assorbimento e la coppia motrice.

Il moto rotatorio soddisfa la seguente equazione del moto, analoga della legge di Newton della meccanica classica:

$$C_m - C_r = \frac{d(J\,\omega)}{dt} = J\frac{d\,\omega}{dt} + \omega\frac{d\,J}{dt} \qquad 24.$$

dove C_r rappresenta la coppia resistente, J il momento di inerzia.

Si vuole sottolineare come, in condizioni stazionarie e quindi esauriti i transitori, la coppia motrice eguagli la coppia resistente. Di conseguenza, in queste condizioni, vi è un rapporto di proporzionalità anche tra la corrente di assorbimento, monitorata nella fase di sperimentazione, e la coppia resistente. Si tratta di uno dei punti cardine della logica utilizzata negli algoritmi diagnostici. L'andamento temporale della corrente rispecchia, a meno di costanti di proporzionalità ed in condizioni stazionarie, l'andamento temporale della resistenza offerta dal carico. Spesso le anomalie si riflettono proprio sulla resistenza offerta dal carico; di conseguenza, monitorando il valore della corrente di assorbimento, è possibile risalire ad eventuali anomalie nel sistema [28].

4.3 Il caso di studio: FTS-300

Grazie alla collaborazione con l'azienda TMC, *Tissue Machinery Company*, è stato possibile procedere con la fondamentale fase di sperimentazione. L'azienda ha messo a completa disposizione un macchinario da loro prodotto: la FTS-300. Si tratta di una macchina impacchettatrice di rotoli di carta; in particolare carta da cucina o carta igienica, in grado di confezionare fino a 200 rotoli al minuto. In Figura 44 si può osservare la parte anteriore della macchina, in Figura 45 la parte posteriore.

Figura 44: Vista frontale della macchina FTS-300.

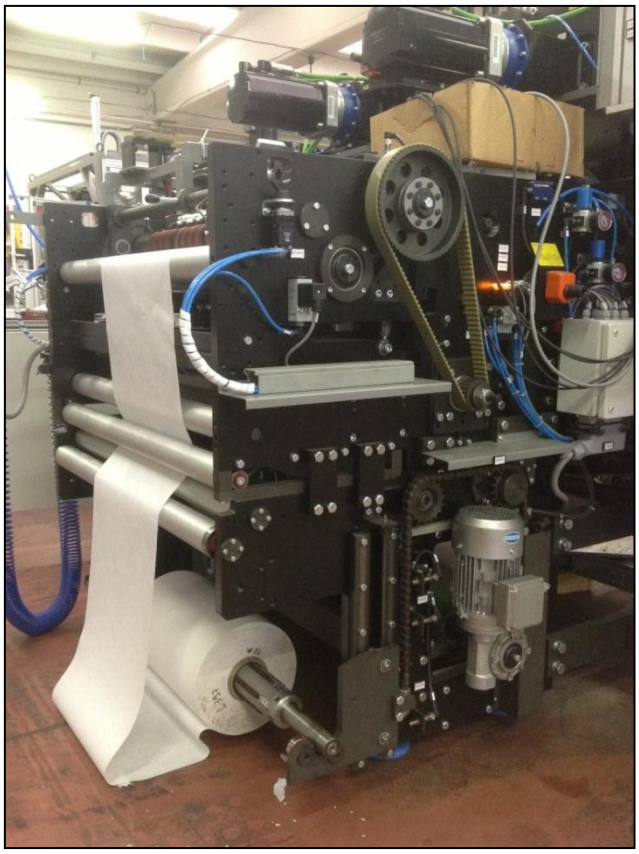

Figura 45: Vista posteriore della macchina FTS-300.

I gruppi principali, visibili nelle due precedenti immagini, sono:

- Alimentazione rotoli
- Alimentazione film
- Dosatore rotoli
- Piattello elevatore
- Piegatore e contro piegatore
- Nastro alimentazione
- Uscita saldante
- Quadro elettrico

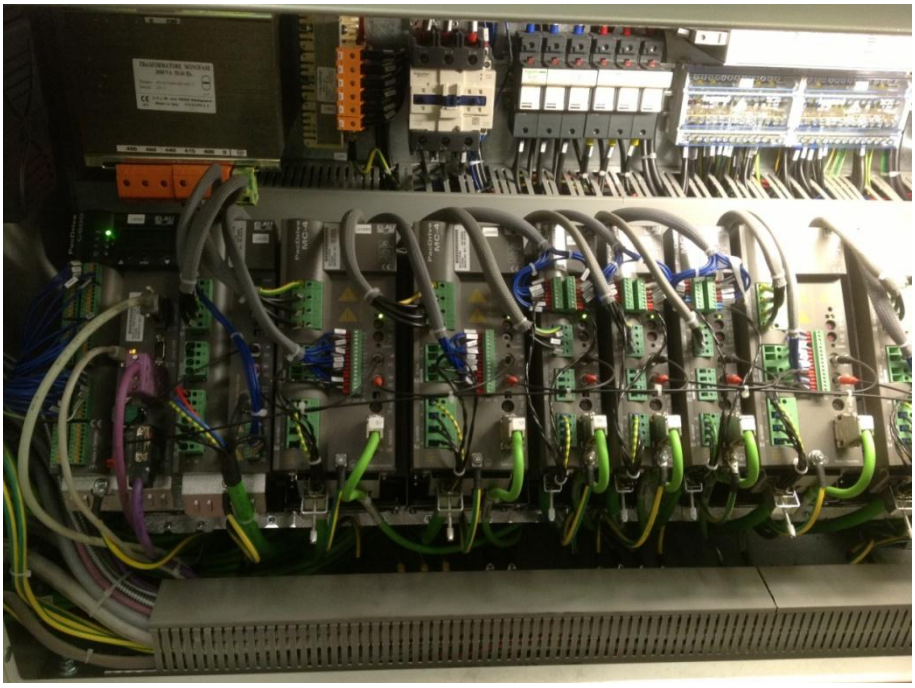

Figura 46: Quadro elettrico e logica di controllo.

Durante la fase di sperimentazione si sono presi in esame il gruppo dosatore, in Figura 47, ed il gruppo piattello elevatore, in Figura 48.

Il dosatore ha il compito di "lanciare" i rotoli all'interno della macchina, rispettando il sincronismo con il piattello elevatore. Quest'ultimo porta al piano superiore della macchina i rotoli per l'impacchettamento. La connessione fra i due gruppi è uno dei punti critici della macchina; il sincronismo, soprattutto ad elevate velocità, non è sempre facile da raggiungere.

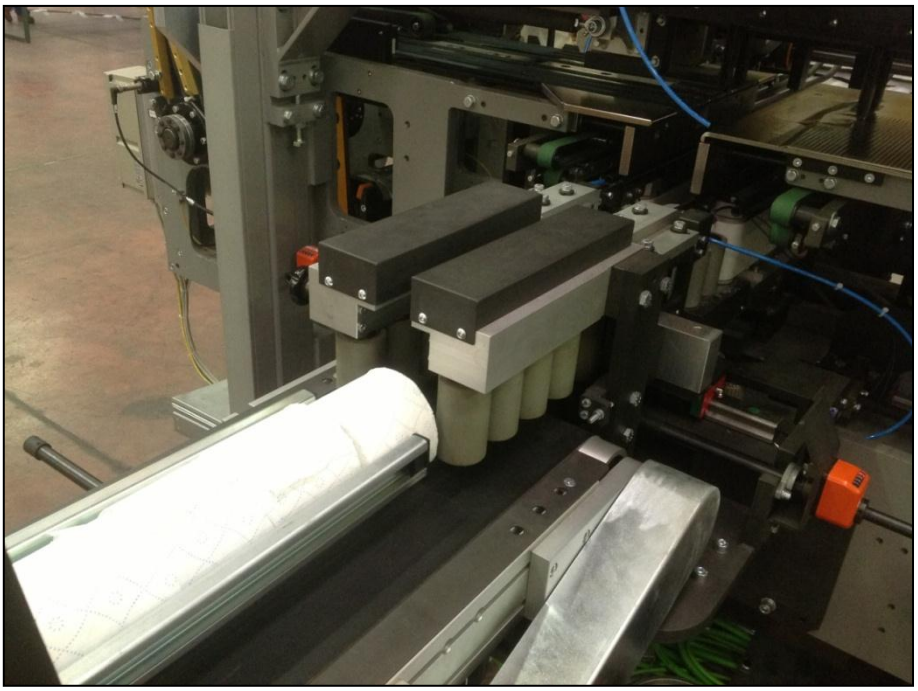

Figura 47: Gruppo dosatore.

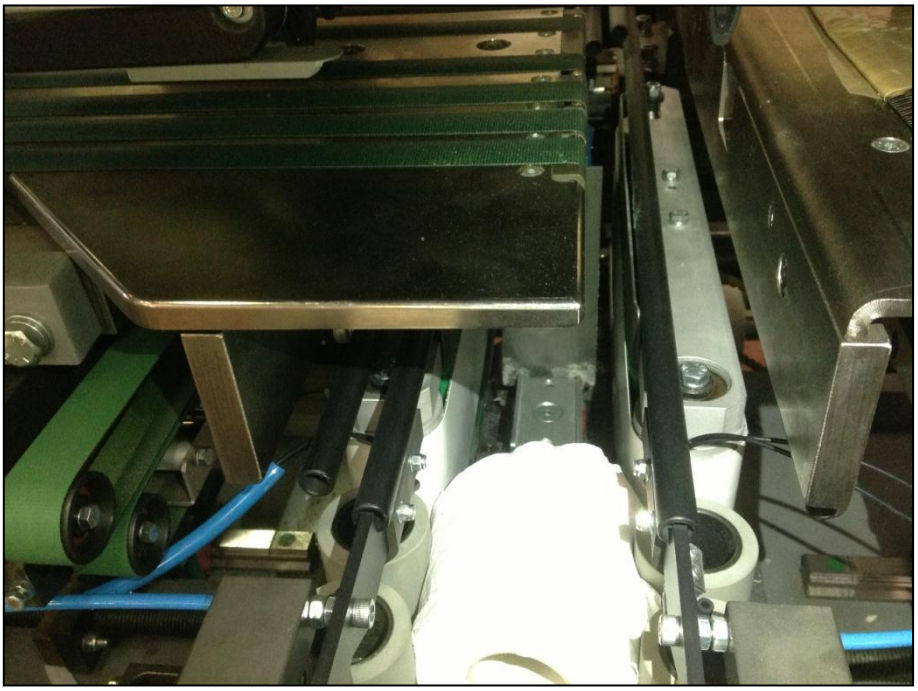

Figura 48: Gruppo piattello elevatore.

Capitolo 5:

Data logging

Un passaggio chiave dell'intero progetto consiste nella lettura e successiva memorizzazione dei valori relativi alle variabili del sistema che si intende monitorare. Tali valori, infatti, sono la materia prima per l'applicazione degli algoritmi diagnostici presentati nei capitoli precedenti.

È stato realizzato un modulo software, in linguaggio Java©, in grado di leggere i dati sfruttando lo standard OPC per poi memorizzarli su un apposito database.

La prima parte si presentava come un compito piuttosto semplice, da svolgere nel corso di qualche giorno. Si è invece rivelato un compito alquanto complicato, richiedendo un impegno di circa due mesi. Il problema più rilevante consisteva nelle intricate interazioni fra numerosi standard ed il sistema operativo Windows©, obbligando spesso ad adottare un approccio di tipo euristico. Un altro punto progettuale delicato è stato il rispetto del sincronismo.

5.1 Lettura dati

5.1.1 Sincronizzazione

L'obiettivo è chiaro: campionare ad istanti regolari i valori di determinate variabili. Il concetto della sincronizzazione, in questo contesto, è vitale poiché nel sistema si trovano ad interagire sistemi differenti: il PLC, i controllori del moto e il computer su cui risiederà il *data-logger*.

Le sincronizzazioni da gestire sono due:

- dal controllore di moto al PLC.
- Dal PLC al *data-logger*

5.1.1.1 Controllore del moto – PLC

Entrambi i sistemi, PLC e controllore del moto, sono di tipo *hard real-time.* Per realizzare un efficiente *data-logger,* è necessario effettuare letture dei parametri con tempi di campionamento minimi, dell'ordine del millisecondo. Per gestire la comunicazione fra tali sistemi è quindi fondamentale un'interfaccia molto performante.

L'interfaccia SERCOS, acronimo di *SErial Real-time COmmunication System* soddisfa i requisiti di determinismo temporale. Una prima versione dello standard è stata definita nel 1991 da un gruppo di aziende *leader* del settore. La prima interfaccia, SERCOS I, sfrutta come mezzo trasmissivo la fibra ottica, utilizza la topologia di rete ad anello e raggiunge il massimo *data rate* di 4Mbit/s. SERCOS II subentra nel 1999 e porta la velocità massima ai 16Mbit/s. Infine la terza versione, SERCOS III risolve i vincoli *hard real-time* sfruttando lo standard Ethernet [29].

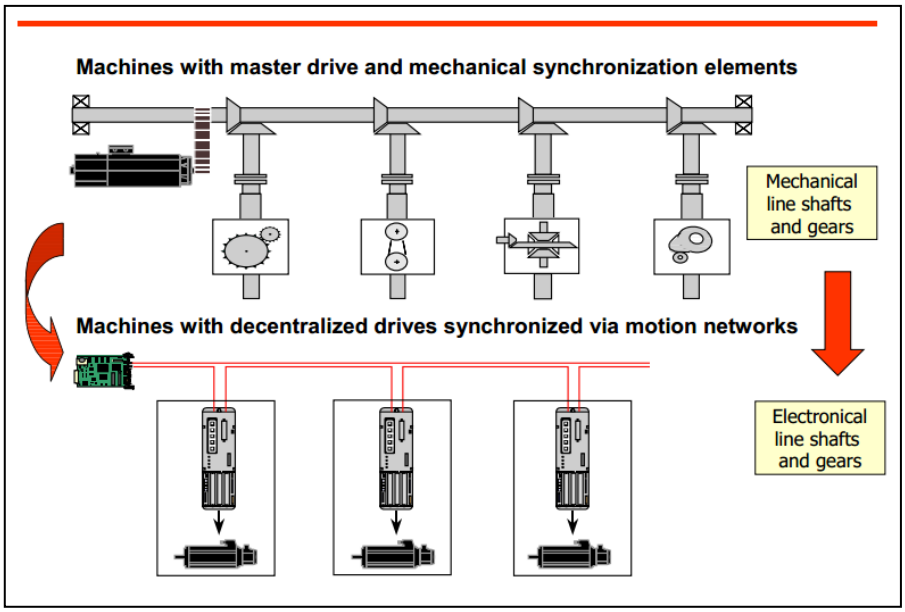

Figura 49: Interfaccia SERCOS.

SERCOS mette a disposizione un canale di servizio che può essere utilizzato per leggere diversi valori relativi agli assi in *real-time*, quali *current, feedback current*, ecc. Il parametro *CycleTime*, all'interno della configurazione dell'interfaccia, può assumere i valori: 1ms, 2ms, 4ms. In ambiente *CoDeSys*© sono previste apposite funzioni per l'utilizzo del canale.

5.1.1.2 PLC – data-logger

In un mondo ideale basterebbe impostare nel *data-logger* realizzato, il tempo di campionamento desiderato. Nel mondo reale, tutto ciò risulta impossibile poiché i programmi realizzati nel linguaggio Java© non vengono eseguiti *real-time*. Si è deciso, perciò, di sfruttare lo strumento *real-time* per eccellenza: il PLC. L'idea è quella di affidare il campionamento ad un *task,* da inserire nel programma preesistente, impostando il tempo di ciclo della sua esecuzione ad un valore pari al tempo di campionamento desiderato. In questo modo è possibile arrivare ad un tempo di campionamento pari ad 1ms. Per poter liberare il *data-logger* dai vincoli temporali troppo stringenti, si è preferito realizzare un *buffer* all'interno del *task.* Viene di seguito presentata una porzione molto semplificata del codice realizzato:

```
index := index + 1;
IF index < length THEN
    ISH01_Doser1_Current_temp[index]:=ISH01_Doser1.Current;
ELSE
    ISH01_Doser1_Current_temp[index]:=ISH01_Doser1.Current;
    ISH01_Doser1_Current :=ISH01_Doser1_Current_temp;
    index := 0;
    SynchTag := SynchTag + 1;
END_IF
```

Nel vettore temporaneo *ISH01_Doser1_Current_temp* ad ogni tempo di ciclo viene scritto, nella posizione definita dall'indice, *index*, il valore attuale della variabile monitorata. Quando l'indice supera la lunghezza preimpostata del vettore, *length,* si scrive l'ultimo valore e poi si fa una copia del vettore temporaneo completo nel vettore di appoggio definitivo *ISH01_Doser1_Current*. A questo punto il *data-logger* può leggere tale vettore, rispettando così i vincoli temporali. Al tempo di ciclo successivo si resetta il vettore temporaneo e si ricomincia il ciclo.

E' da sottolineare l'importanza di un ulteriore indice: *SynchTag*. L'indice viene incrementato ad ogni nuovo vettore definitivo, segnalando quindi la presenza di nuovi dati, pronti per la lettura. Il *data-logger*, semplicemente controllando il valore di tale indice, sarà in grado di evitare doppie letture o di segnalare eventuali dati persi. In questo modo si svincola completamente il *data-logger* dal PLC.

5.1.2 OPC: *Open Platform Communications*

Risolto il problema della sincronizzazione, occorre superare l'ostacolo della diversa "lingua" parlata dai due sistemi: per metterli in comunicazione occorre adottare regole comuni. In campo industriale è stato adottato lo standard OPC.

OPC, acronimo di *Object Linking and Embedding (OLE) for Process Control*, è uno standard sviluppato nel 1996 da un gruppo di industrie del ramo dell'automazione e di fornitori software. Lo standard fornisce le specifiche di comunicazione di dati *real-time* tra dispositivi di diversa provenienza manifatturiera. Il gruppo di industrie, detto *OPC Foundation*©, ha poi ufficialmente ribattezzato, nel novembre 2011, l'acronimo di OPC in *Open Platform Communications*. Infatti lo standard è stato progettato con lo scopo di mettere a disposizione un "ponte" tra applicazioni basate su Windows© e hardware specifico per il controllo di processo.

Lo standard prevede un'architettura *Client-Server* che consente alle applicazioni di controllo l'accesso in tempo reale alle informazioni che provengono dal PLC.

Le specifiche OPC sono basate sulle tecnologia OLE sviluppata da Microsoft© per i sistemi operativi Windows©, meglio conosciuta come *Component Object Model*, o COM. Per facilitare l'interoperabilità, vengono definiti un insieme standard di oggetti, interfacce e metodi da utilizzare in applicazioni di automazione. La specifica principale è OPC *Data Access* (OPC DA), che descrive l'interfaccia di accesso ai dati ed è utilizzata per leggere e scrivere dati *real-time*.

Lo standard definisce dei metodi che accedono a determinati campi dei dati, indipendentemente dalla tipologia della sorgente del dato stesso [30].

5.1.2.1 Architettura *Client-Server*

Lo standard, per garantire la fondamentale caratteristica di interoperabilità, prevede un sistema di tipo *Client-Server*. Tale sistema prevede la connessione di un'applicazione, detta *applicazione client*, ad un'altra applicazione detta *applicazione server*, attraverso l'istanziamento dell'interfaccia. Le due applicazioni, *OPC-client* e *OPC-server*, possono risiedere o meno sullo stesso computer.

Il *client OPC* è un programma addetto alla visualizzazione delle variabili, dette *item* o *tag*, che si intende monitorare. Inoltre consente la memorizzazione dei valori relativi alle variabili di interesse.

Il *server OPC*, invece, è un programma eseguibile che parte automaticamente durante la connessione fra client e PLC. Il server, grazie al *Gateway* è in grado di fornire al client tutte le variabili, ed i relativi valori, disponibili sul PLC. Sarà compito del client impostare le variabili che si intendono monitorare e con quale frequenza, *update rate*, aggiornarne i valori [31] [32].

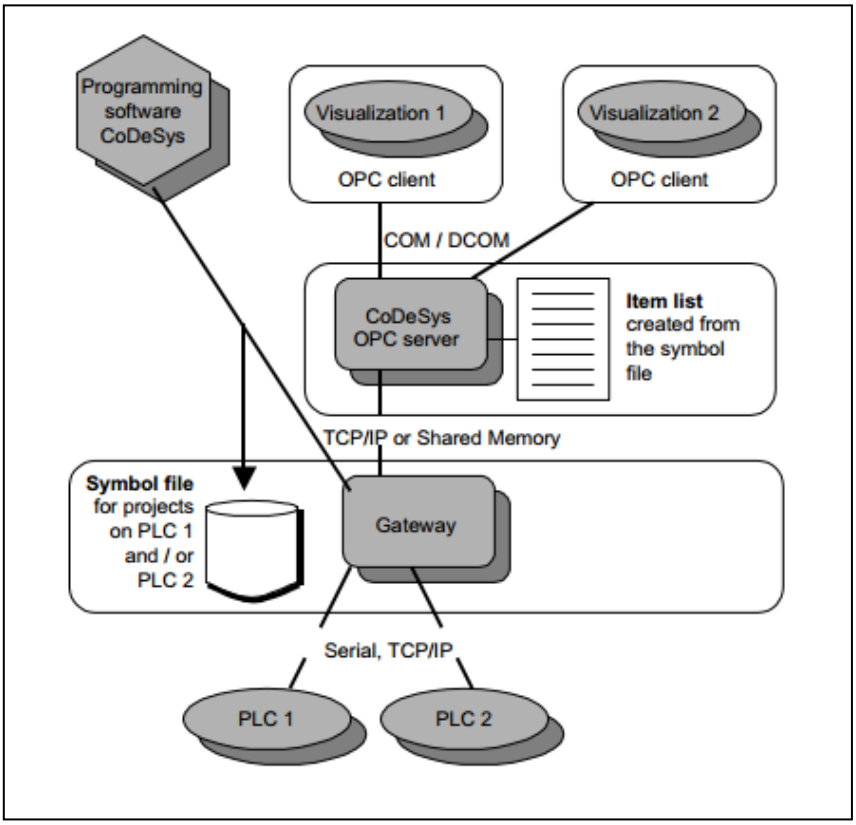

Figura 50:Architettura di comunicazione.

Gli *item* possono essere organizzati nei cosiddetti *groups*. Ciascun *group* possiederà una propria frequenza di campionamento.

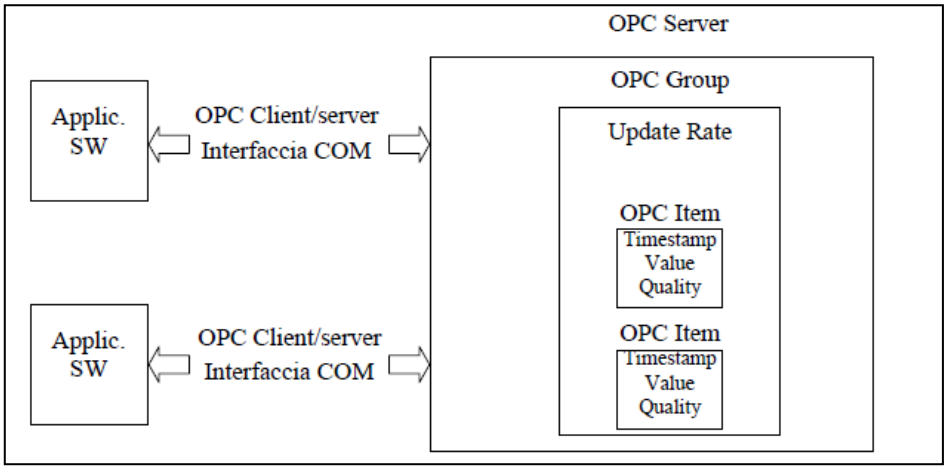

Figura 51: OPC Client/Server

5.2 Memorizzazione dati

Il *data-logger* raccoglie una mole enorme di dati. Le informazioni devono poi essere aggregate e successivamente memorizzate in un archivio. Per questo motivo è stato progettato, insieme ai colleghi maggiormente preparati in ambito informatico, un *database* basato su modello relazionale. La scelta progettuale è ricaduta su *MySQL*©*, database open-source* sviluppato da *Oracle*©.

Il filtro è stato ideato, anche se non ancora implementato, per rendere possibili le operazioni di aggregazione. L'aggregazione dei dati è un punto essenziale nel progetto. Basti pensare che si è stimata, in maniera grossolana, una quantità di dati giornaliera dell'ordine di qualche *gigabyte*.

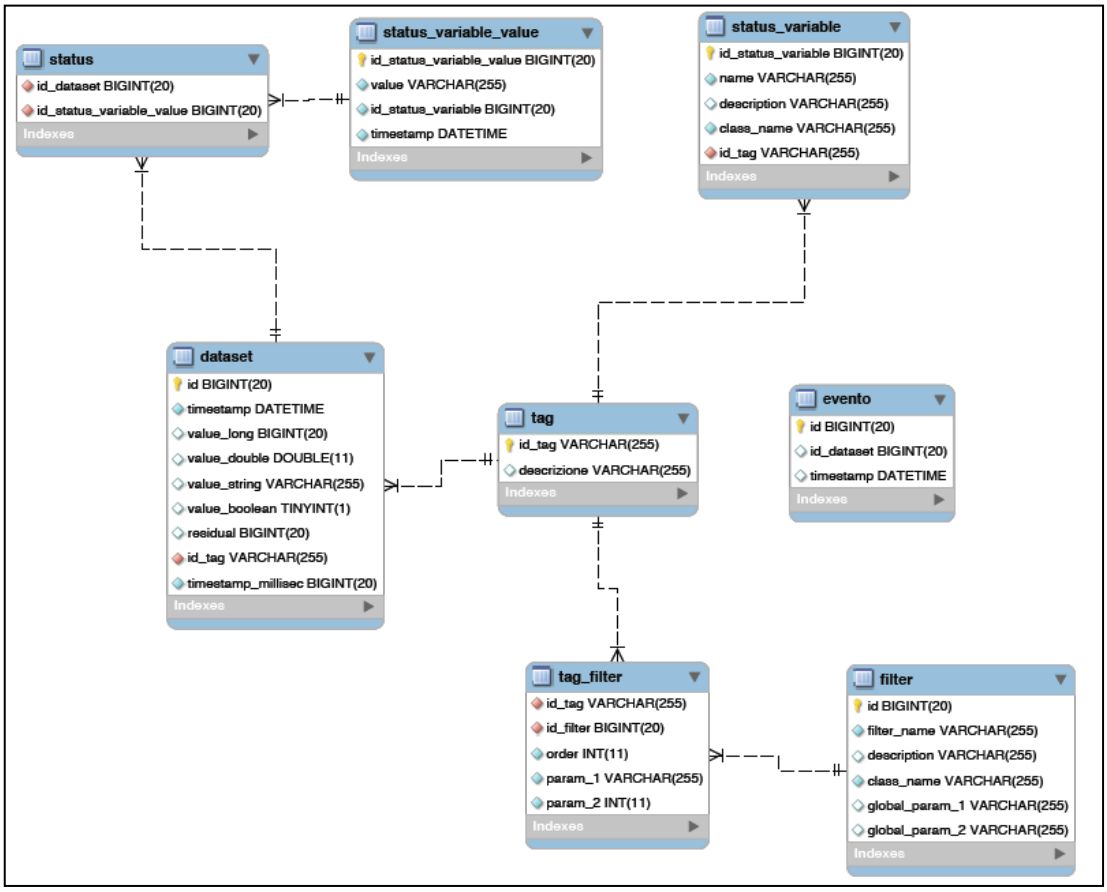

Figura 52: Struttura del database.

Capitolo 6:

Risultati ottenuti

Per una corretta e chiara interpretazione dei dati rilevati si è passati alla fase di visualizzazione dei dati stessi. I dati sono stati importati dal *database* (il codice in Appendice D), elaborati ed infine graficati attraverso funzioni appositamente progettate in ambiente Matlab©.

Le prime prove consistono semplicemente in un monitoraggio della corrente del gruppo dosatore nelle seguenti condizioni:

- condizione iniziale: macchina ferma.
- macchina "a vuoto", cioè in assenza del prodotto.
- velocità a regime: 50 pacchi al minuto.

In Figura 53 sono stati riportati i valori istantanei di tale corrente:

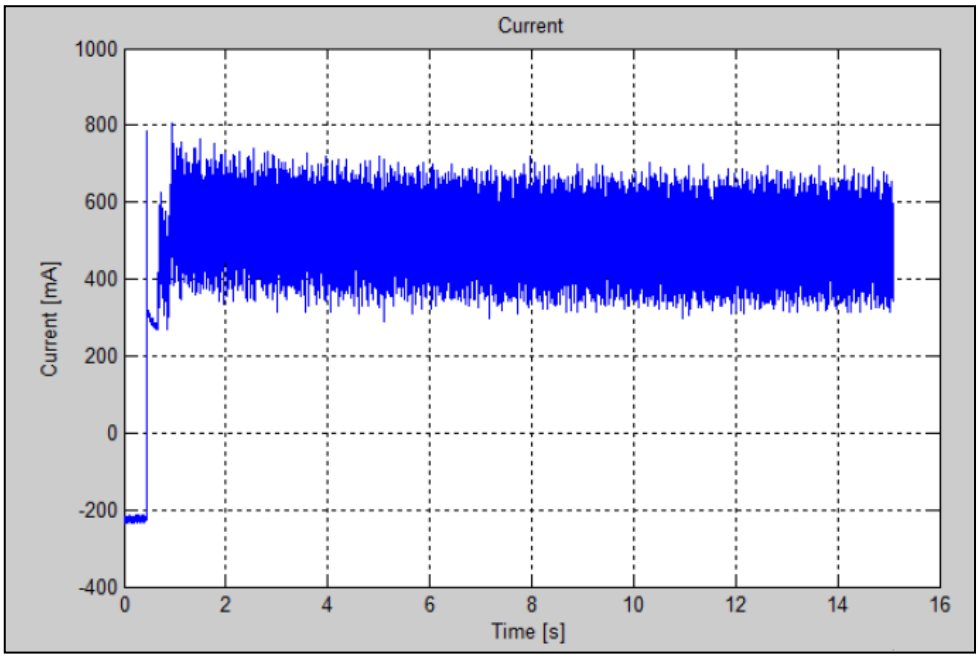

Figura 53: Valori istantanei di corrente del dosatore.

Si è pensato di analizzare il *ripple* di corrente, valutandone il contenuto spettrale a diverse velocità. Si può osservare in Figura 54, con velocità 50 pacchi/minuto, due componenti fondamentali, rispettivamente a 1.45 Hz e 13.45 Hz. Tutte le altre componenti sono armoniche, cioè si presentano a frequenze multiple della fondamentale (1.45 Hz).

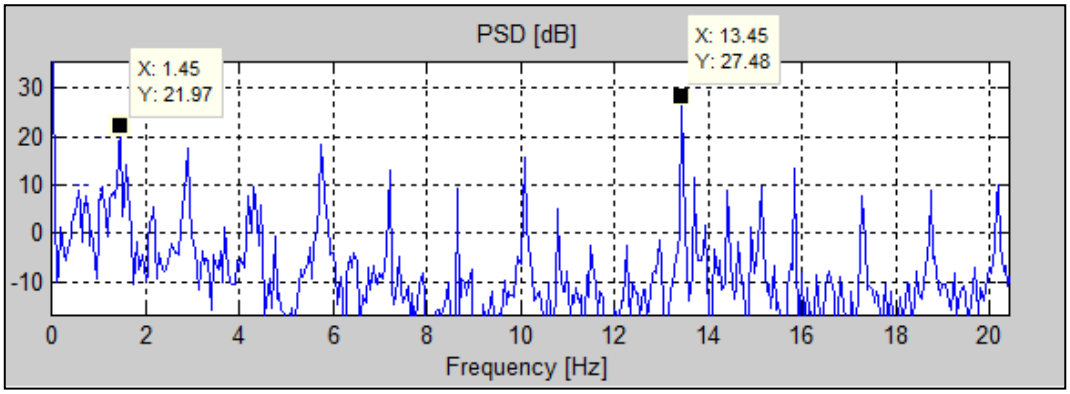

Figura 54: Densità spettrale di potenza della corrente del dosatore a velocità 50 pacchi/minuto.

In Figura 55, invece, è presente la densità spettrale di potenza della corrente riferita al funzionamento della macchina a 80 pacchi al minuto. Si può osservare il diverso contenuto spettrale: in questo caso si presenta un'unica componente fondamentale a 1.15 Hz e le sue armoniche.

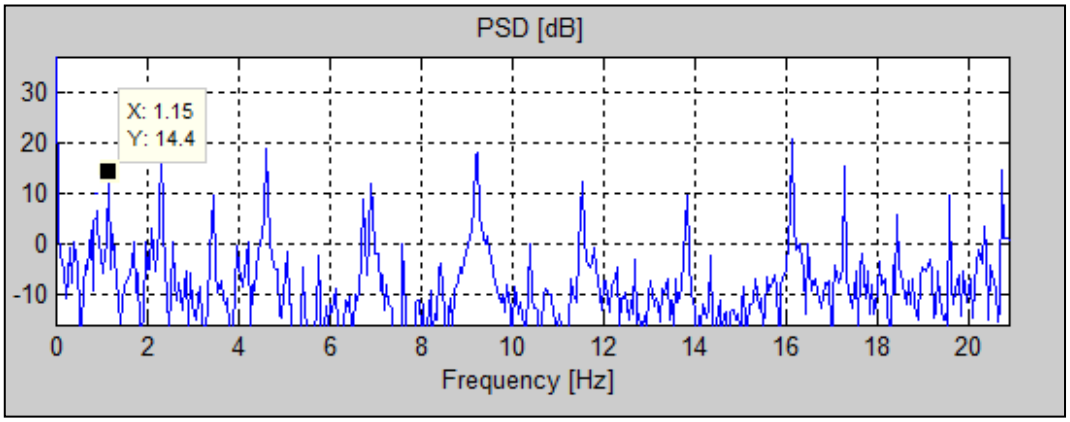

Figura 55: Densità spettrale di potenza della corrente del dosatore a velocità 80 pacchi/minuto.

Dalla Figura 53 risulta in maniera evidente l'elevata ampiezza del *ripple* di corrente. Per limitare questo effetto si è implementato un filtraggio: in questo modo, da una parte, si è semplificata la comprensione del grafico e dall'altra si è alleggerita la mole di dati da gestire. Il filtraggio è stato implementato attraverso una semplice media aritmetica, su una finestra di campioni di larghezza configurabile; in Figura 56 è rappresentata la versione filtrata, con una finestra di 1000 campioni. Si osserva, dopo alcuni secondi di assestamento, un valore pressoché costante della corrente intorno ai 500mA. D'ora in poi tutti i grafici di corrente presenteranno i valori filtrati.

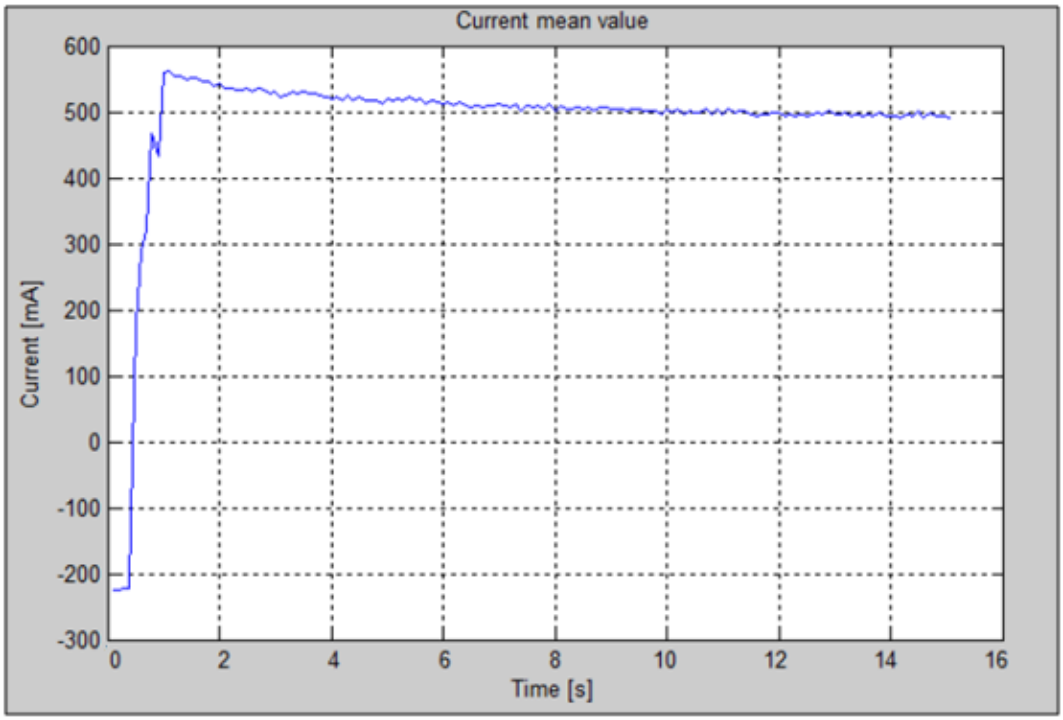

Figura 56: Valori filtrati di corrente del dosatore.

Come seconda prova, in Figura 57, si è voluto mettere a confronto, in blu, la corrente del dosatore con la macchina a vuoto, e in rosso, in presenza del prodotto. Questo risultato anche se apparentemente banale è di grande interesse, poiché sottolinea la sensibilità della corrente del motore alle resistenze esterne. Infatti si registra una variazione della corrente che passa dai 500mA nel caso "a vuoto", ai 570mA nel caso "con prodotto": una differenza notevole dell'ordine del 12%.

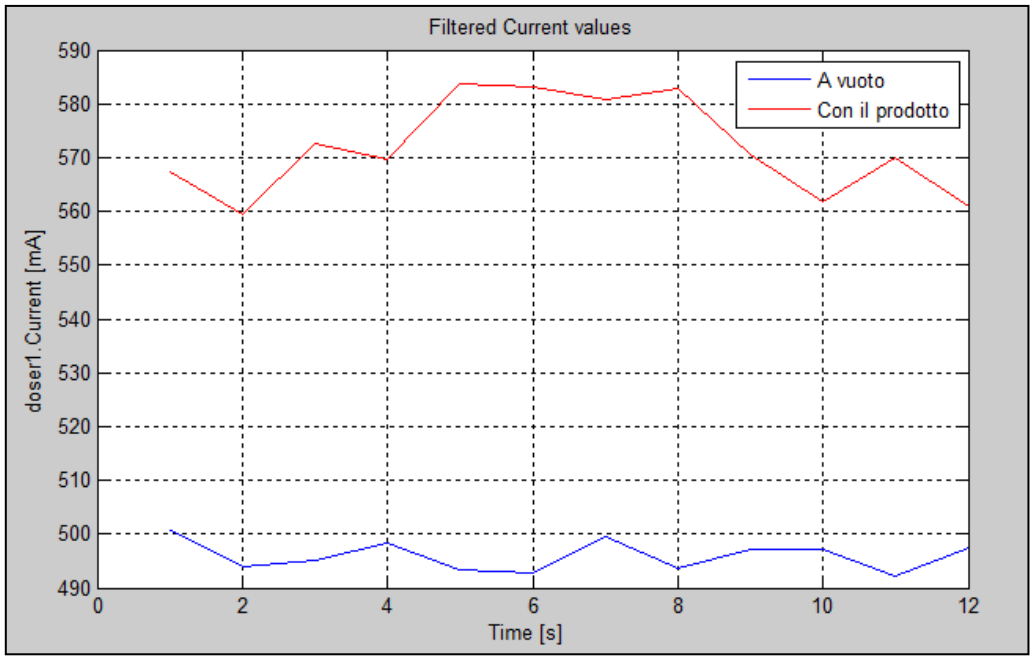

Figura 57: Confronto dei valori filtrati di corrente in presenza ed assenza di prodotto.

Poi si passati alla simulazione di un guasto; la rottura della cinghia. Si è quindi tolta una cinghia di trasmissione, dimezzando così il carico inerziale del motore. Ovviamente anche la corrente si è ridotta della metà, come evidente in Figura 58:

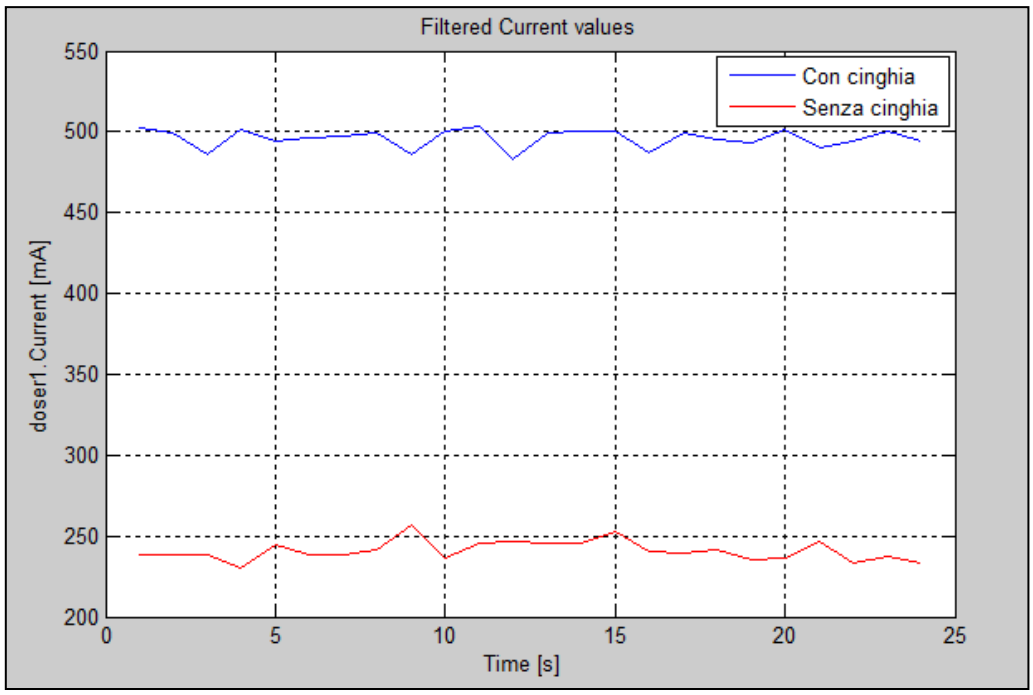

Figura 58: Confronto corrente del dosatore in presenza ed assenza di una cinghia di trasmissione.

In Figura 59 si può osservare la parte superiore del gruppo dosatore. In particolare dove si notano gli ingranaggi, è stato smontato il *carter* di una delle due file di cilindri. Il tentativo era quello di simulare, introducendo un semplice fazzoletto di carta, la presenza di sporcizia negli ingranaggi, situazione molto comune e di difficile rilevamento.

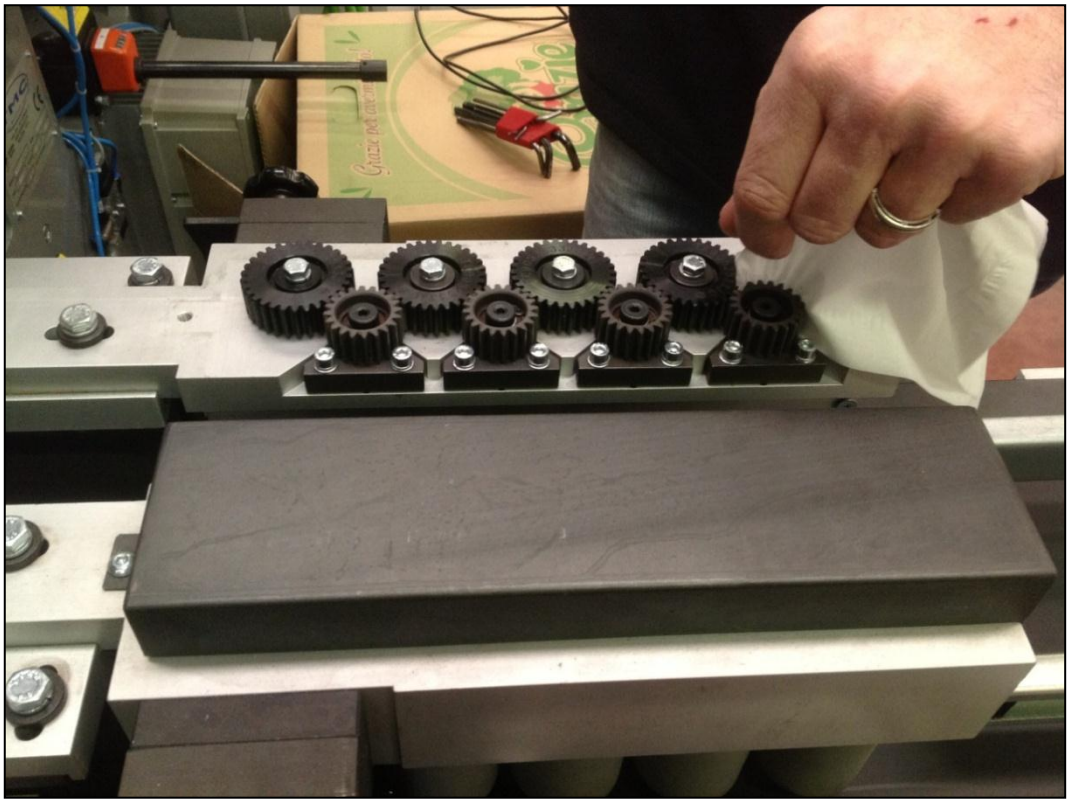

Figura 59: Simulazione *fault*: attrito anomalo causato da sporcizia accumulata negli ingranaggi.

Il risultato è rilevante; in corrispondenza dell'istante in cui viene inserita la carta, intorno all'undicesimo secondo, si registrano picchi di corrente con variazioni di oltre il 50% rispetto al suo valore standard come evidenziato in Figura 60.

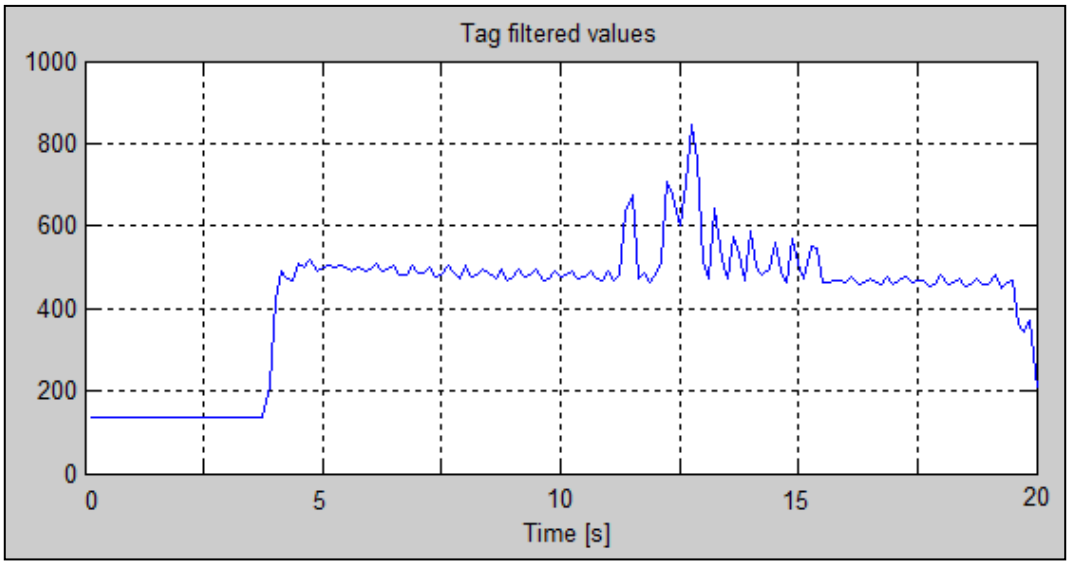

Figura 60: Corrente del dosatore in presenza di attrito anomalo causato da sporcizia accumulata negli ingranaggi.

Si è successivamente simulato un attrito anomalo esterno, mettendo a contatto un pezzo di polistirolo con i cilindri del dosatore, come rappresentato in Figura 61.

Figura 61: Simulazione *fault*: attrito anomalo causato da evento esterno.

Il contatto è stato effettuato fra i 18 e i 32 secondi. Come evidente in Figura 62 durante questo intervallo temporale si è verificato un incremento di corrente del 16% anche a fronte di una debole interferenza.

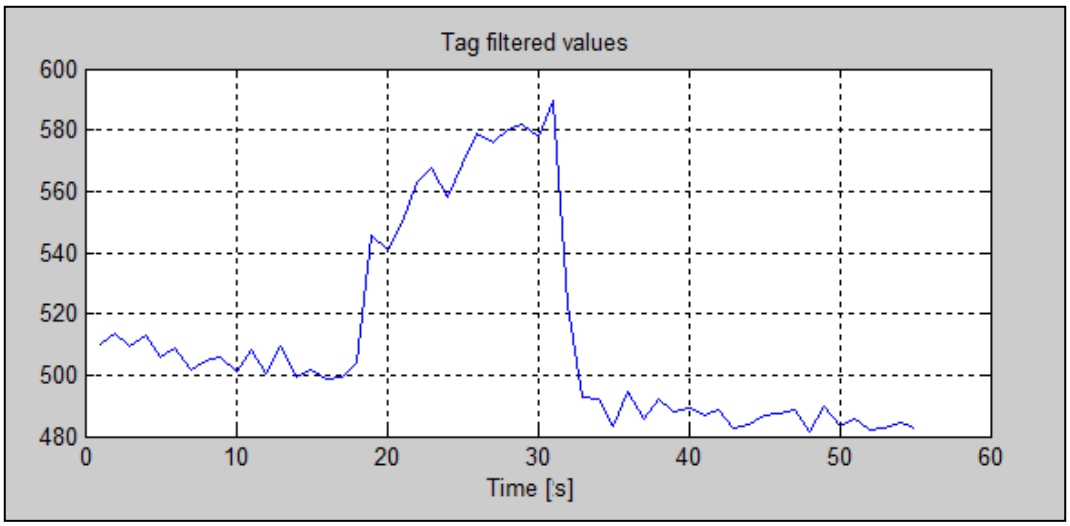

Figura 62: Corrente del dosatore in presenza di attrito anomale causato da evento esterno.

In ultima analisi si è posta l'attenzione sul gruppo del piattello elevatore. Considerando la dinamica della piattello più interessante rispetto a quella più monotona dei cilindri del dosatore, si è scelto di allargare il campo delle variabili da monitorare, inserendo anche la velocità del piattello ed il suo *set-point.* Il *set-point*, identificato dal prefisso "*Ref*", rappresenta il valore fissato dalla logica di controllo che deve essere raggiunto dal sistema di attuazione. Rappresenta, in sostanza, il comando da rispettare.

In Figura 63 si effettua, in condizioni nominali, il confronto fra la corrente del piattello elevatore, la sua velocità ed il rispettivo *set-point.*

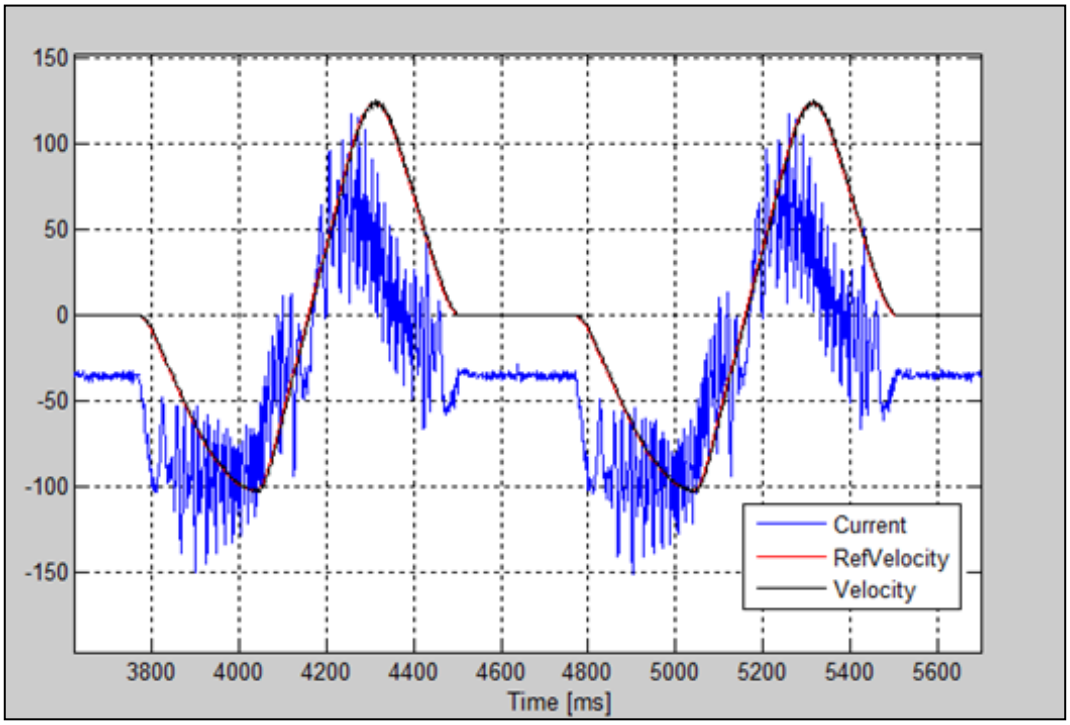

Figura 63: Confronto tra corrente, velocità e suo *set-point* nel piattello elevatore in condizioni nominali.

Chiaramente i segnali presentano una caratteristica periodica, determinata dalla velocità impostata nella macchina. La corrente, per facilitare la visualizzazione, è stata ridotta di un fattore dieci. Si presenta nuovamente il fenomeno del *ripple* di corrente, anche se con un ampiezza inferiore. Apparentemente *RefVelocity* e *Velocity* coincidono perfettamente; in realtà attraverso lo zoom si è trovata una latenza di circa 6ms. Questo intervallo temporale rappresenta dunque il tempo trascorso tra la ricezione del comando e la sua esecuzione.

In seguito si è procurato un inceppo al piattello in modo da studiare il comportamento dei segnali in quel frangente. L'inceppo si verifica nel momento in cui il piattello, in fase di discesa, si incastra con il rotolo che, a causa di un errore nella fase di lancio, si trova sotto al piattello stesso. Il risultato, molto interessante, è presentato in Figura 64.

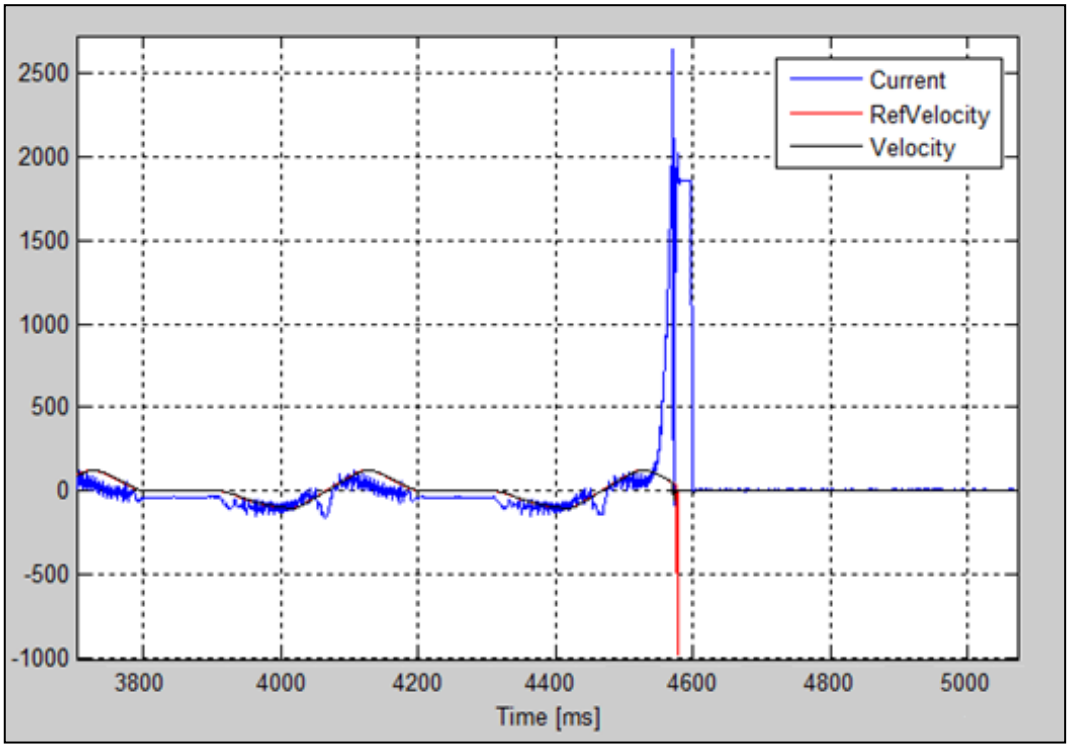

Figura 64: Confronto tra corrente, velocità e suo *set-point* nel piattello elevatore in caso di inceppo.

Il valore di corrente, nel momento in cui il piattello, in discesa, entra in contatto con il rotolo sottostante, aumenta in maniera vertiginosa.

Per osservare meglio cosa accade negli istanti in cui avviene l'inceppo si presenta lo zoom in Figura 65.

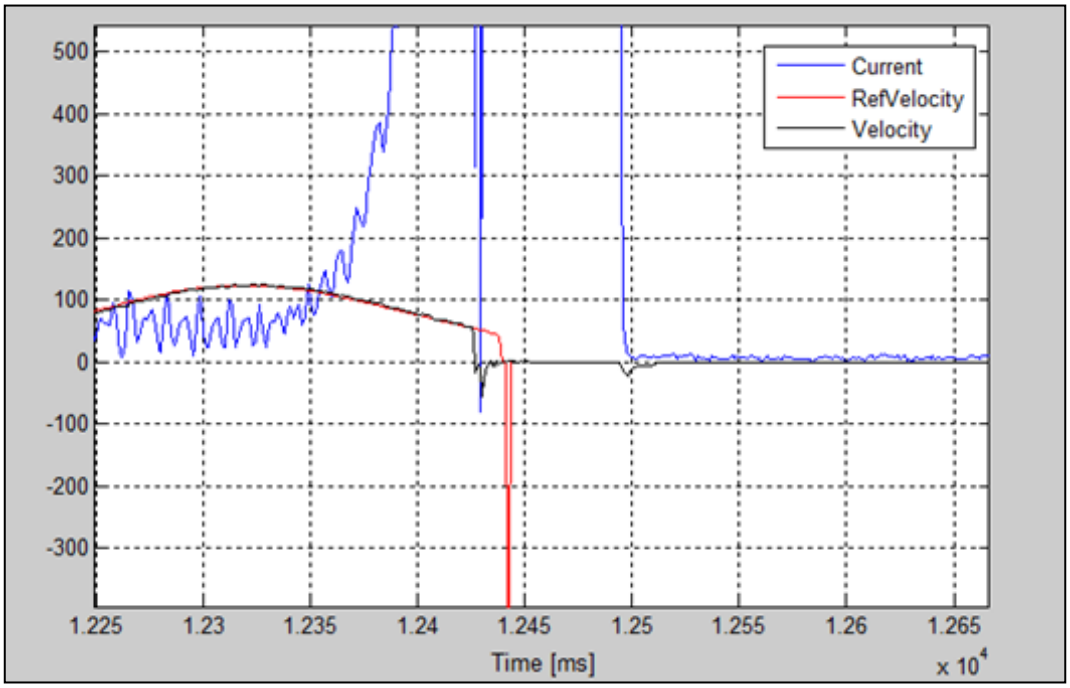

Figura 65: Zoom del confronto tra corrente, velocità e suo *set-point* nel piattello elevatore in caso di inceppo.

Il punto più interessante da sottolineare è la notevole latenza temporale del sistema di controllo: infatti dall'aumento sconsiderato di corrente al comando di riduzione di velocità, *RefVelocity*, trascorrono quasi 100 ms. Occorre, quindi in futuro, concentrarsi sulla diminuzione di questa latenza in modo da ridurre i possibili danni causati dall'inceppo.

Infine, data la caratteristica periodica della corrente del piattello, si è pensato di utilizzare la rete neurale e valutarne le prestazioni in questa applicazione. Il *dataset* della corrente, riferito al caso nominale, è stato utilizzato per la fase di *training* della rete. In Figura 66 il risultato della fase di *test*, in condizioni nominali, di una semplice rete composta da 10 percettroni e 10 linee di ritardo.

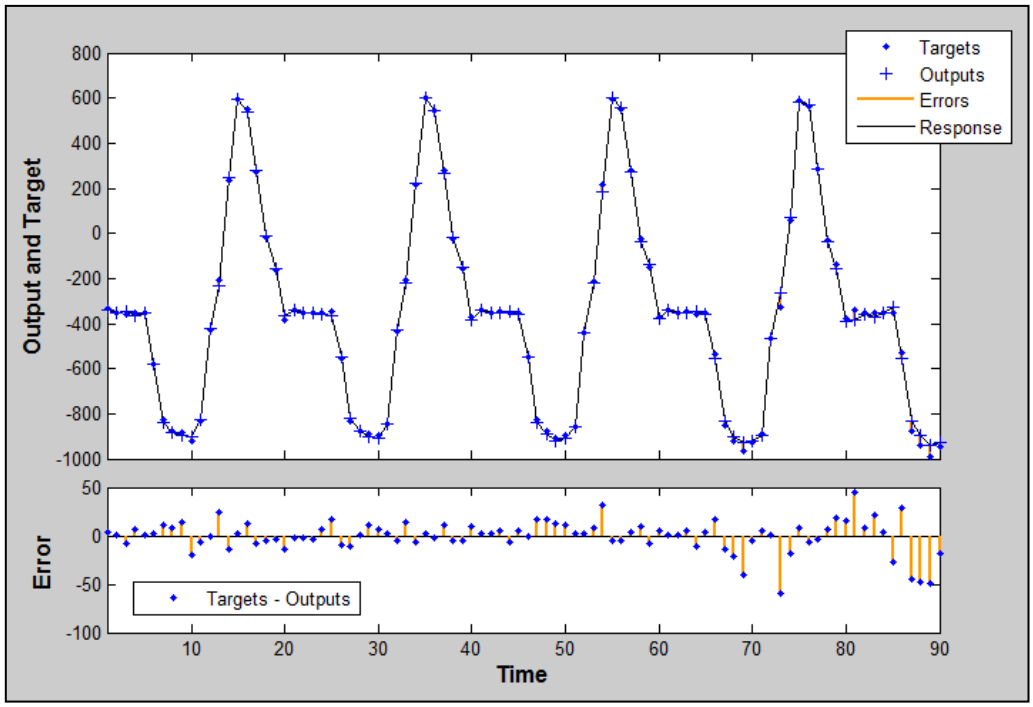

Figura 66: Test della rete neurale, in condizioni nominali: dataset riferito alla corrente del piattello elevatore.

In Figura 67, il *test* della stessa rete, nel caso di inceppamento. È più che evidente l'ampiezza dei residui prodotti dalla rete e quindi il corretto rilevamento del *fault*.

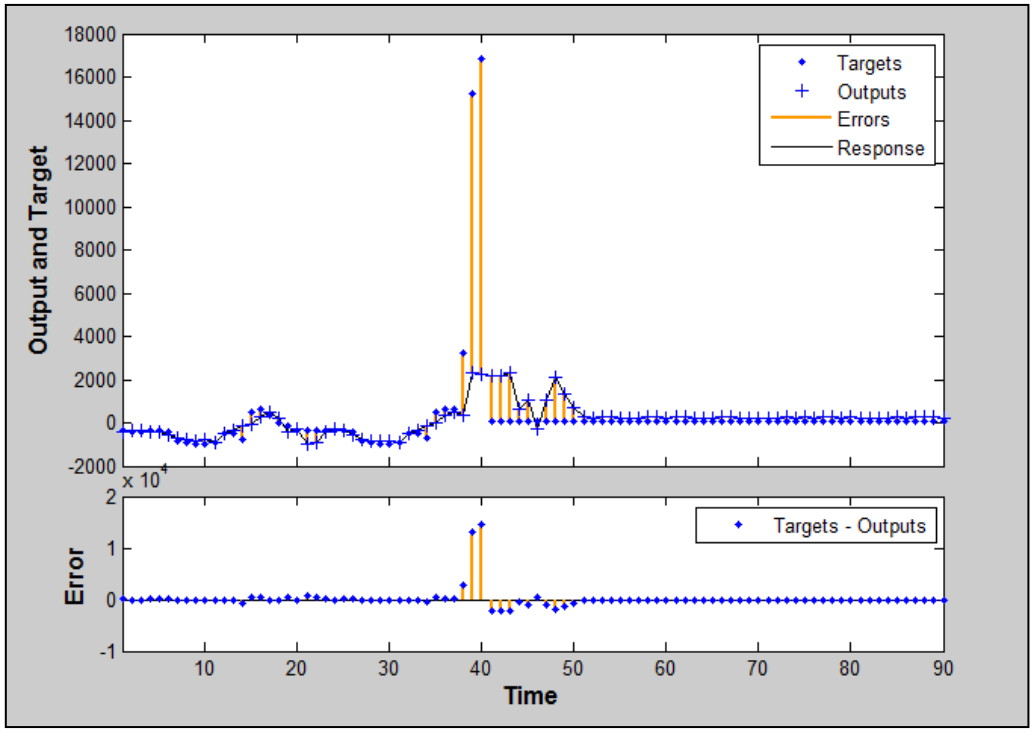

Figura 67: Test della rete neurale, in caso di inceppamento: dataset riferito alla corrente del piattello elevatore.

Capitolo 7:

Conclusioni e sviluppi futuri

La fase di ricerca ha messo in evidenza numerosi algoritmi da poter utilizzare nell'attività di *Fault Detection and Isolation*. Sono stati individuati strumenti molto semplici, ma molto efficaci, come l'approccio statistico e il "fuori-range", e strumenti più complessi e dalle potenzialità ancora da studiare in questo campo, quali la rete neurale e l'analisi spettrale. Come evidenziato nel capitolo 6, gli algoritmi implementati sono in grado di rilevare i seguenti guasti e malfunzionamenti:

- attrito anomalo causato da sporcizia accumulata negli ingranaggi del gruppo dosatore.
- attrito anomalo causato da evento esterno al gruppo dosatore.
- rottura della cinghia di trasmissione del gruppo dosatore.
- inceppo del piattello elevatore.

Le reti neurali, come visto nel capitolo 2, presentano potenzialità davvero interessanti e, di conseguenza, meritano ampio spazio di ricerca. Nell'attività svolta in azienda ci si è concentrati sullo studio del *training* della rete ad una sola variabile. Uno sviluppo considerevole del progetto consisterebbe nella ricerca dell'utilizzo di *dataset* a più variabili, poiché ciò consentirebbe di fruire delle ulteriori informazioni celate nelle loro reciproche relazioni.

Un ulteriore passo in avanti sarebbe l'impiego di sensori aggiuntivi, come ad esempio gli accelerometri, per due motivazioni principali. Prima di tutto per rendere i *dataset*, da passare alle reti neurali, ancora più rappresentativi del funzionamento della macchina. Inoltre si potrebbero applicare altri approcci algoritmici, come ad esempio quello dell'analisi spettrale delle vibrazioni.

Fino ad ora gli algoritmi sono stati applicati al gruppo dosatore e piattello elevatore. Inoltre l'applicazione è avvenuta solo a determinate velocità di funzionamento e in un unico tipo di formato del prodotto.

Per una corretta generalizzazione dei concetti, sarà importante prima di tutto, estendere l'applicazione degli algoritmi ai restanti gruppi della macchina impacchettatrice. Inoltre occorrerà trovare una relazione che renda indipendente l'algoritmo dalla modalità di funzionamento (velocità, formato…).

Il tempestivo rilevamento del *fault* è di fondamentale importanza poiché consente la riduzione dei tempi di intervento e quindi dannosi fermi di produzione, con evidenti vantaggi per l'azienda. Ancor più considerevole sarebbe il rilevamento di un guasto o malfunzionamento in via predittiva. Come evidenziato nel capitolo precedente, infatti, vi sono intervalli temporali in cui sarebbe possibile intervenire per ridurre il più possibile gli effetti del *fault*, o in alcuni casi addirittura prevenirli. Questo consentirebbe di limitare, o perfino evitare, i danni provocati dal *fault* alla macchina automatica. Uno dei prossimi sviluppi è l'interfacciamento degli algoritmi diagnostici con la logica del sistema di controllo, in maniera tale da condizionare il comportamento della macchina a fronte di anomalie rilevate dal software diagnostico, ed effettuare eventuali operazioni di *recovery*.

Il prodotto software, progettato per l'interfacciamento con la macchina automatica, e la successiva lettura e archiviazione dei dati, si è rivelato molto interessante per le aziende del settore. Il *data-logger* ha, infatti, la capacità di evidenziare, agli operatori specializzati, un punto di vista della macchina spesso demandato ai soli programmatori del PLC. Si tratta di uno strumento in grado di mettere in contatto la fase di progettazione del programma di funzionamento della macchina e le fasi di manutenzione e riparazione; questo favorirebbe un arricchimento reciproco dei due settori.

Attualmente la visualizzazione dei dati rilevati è delegata esternamente al software. Il prossimo passo, già in fase di implementazione, prevede un modulo software adibito alla graficazione *run-time* dei dati monitorati.

Appendice A:

Prediction

Training della rete

Per la fase di training è stato implementato il seguente *script* di Matlab:

```
function net=nn_train(x,n)
    % Design network
    feedbackDelays = 1:10; % window
    hiddenLayerSize = 10; % neurons number
    u=floor(length(x)/n);
    tags=zeros(1,u);
    for j=1:1:(u)
        tags(j)=mean(x(((n*j)-n+1):(n*j)));
    end
    dim=size(tags);
    n_tags=dim(1);
    n_samples_input=dim(2);
    MAX=max(tags);
    MIN=min(tags);
    ones = zeros(1,n_samples_input)+1;
    targetSeries = mat2cell(tags, [n_tags] , ones);

    % Create a Nonlinear Autoregressive Network
    net = narnet(feedbackDelays,hiddenLayerSize);

    % Choose Feedback Pre/Post-Processing Functions
    net.inputs{1}.processFcns = {'mapminmax'};
    net.layers{1}.transferFcn = 'logsig';
    net.layers{2}.transferFcn = 'purelin';
    % Prepare the Data for Training and Simulation
    [inputs,inputStates,layerStates,targets] =
preparets(net,{},{},targetSeries);
    % Setup Division of Data for Training, Validation, Testing
    net.divideFcn = 'divideblock';  % Divide data
    net.divideMode = 'time';  % Divide up every value
```

```
    net.divideParam.trainRatio = 85/100;
    net.divideParam.valRatio = 15/100;
    net.divideParam.testRatio = 0/100;

    % Choose a Training Function
    net.trainFcn = 'trainlm';  % Levenberg-Marquardt

    % Choose a Performance Function
    net.performFcn='mse';  % mean square error

    % Choose Plot Functions
    net.plotFcns = {'plotperform','plottrainstate',
'plotresponse','ploterrcorr', 'plotinerrcorr'};

    % Point Stop train
    net.trainParam.min_grad=1e-6;net.trainParam.goal=1e-6;
    net.trainParam.max_fail=5;net.trainParam.epochs=300;
    net.trainParam.mu_max=1e8; net.trainParam.mu=1;
     net.trainParam.mu_dec=0.8; net.trainParam.mu_inc=1.2;

    % Train the Network
    [net,tr] =
train(net,inputs,targets,inputStates,layerStates);

    % Test the Network
    outputs = net(inputs,inputStates,layerStates);
    errors = gsubtract(targets,outputs);
    performance = perform(net,targets,outputs);

    % Recalculate Training, Validation and Test Performance
    trainTargets = gmultiply(targets,tr.trainMask);
    valTargets = gmultiply(targets,tr.valMask);
    testTargets = gmultiply(targets,tr.testMask);
    trainPerformance = perform(net,trainTargets,outputs);
    valPerformance = perform(net,valTargets,outputs);
    testPerformance = perform(net,testTargets,outputs);

    % Plots
    figure, plotperform(tr)
    figure, plotresponse(inputs,outputs);
end
```

Test della rete

All'interno del software diagnostico è implementata la sola fase di test, sviluppata in Java. In seguito la parte principale del codice:

```
public NeuralNetwork(String network) throws IOException{
        this.network=network;
        String path =
"C:\\Users\\Marco\\Dropbox\\Materiale\\Artificial Neural
Networks\\NN_Matlab\\"+network+"\\";
        this.n_neurons=Import(path+"n_neurons.csv");
        this.delays=Import(path+"delays.csv");
        this.n_tags=Import(path+"n_tags.csv");
        this.IW=
MatrixIO.loadCSV(path+"IW.csv",n_neurons,delays*n_tags);
        this.LW=
MatrixIO.loadCSV(path+"LW.csv",n_tags,n_neurons);
        this.IB= MatrixIO.loadCSV(path+"IB.csv",n_neurons,1);
        this.LB= MatrixIO.loadCSV(path+"LB.csv",n_tags,1);
        setMin();
        setMax();
        this.alpha=2/(this.max-this.min);
    }

    public double[] evaluate(double[] a) {
        long time = java.lang.System.currentTimeMillis();
        DenseMatrix64F interm,residual,o,out,output;
        tags=fromDouble(a);
        int n_samples=tags.getNumCols();
        out=new DenseMatrix64F(1,n_samples);
        output=new DenseMatrix64F(1,n_samples);
        residual=new DenseMatrix64F(1,n_samples);
        buffer = new SimpleMatrix(1,delays);
        DenseMatrix64F array=norm(true,tags);
        t_ = SimpleMatrix.wrap(array);
        for (int k=0;k<n_samples-delays;k++){
            windowing(k);
            interm=mult(buffer,transp(IW));
            interm=sum(interm,transp(IB));
            interm=sigmoid(interm);
            o=mult(interm,transp(LW));
            o=sum(o,LB);
```

```
                out.set(k+this.delays,o.get(0));
            }
            output=norm(false,out);
            CommonOps.sub(this.tags,output,residual);
            double[] c=ResidualToDouble(residual);
           long timefin = java.lang.System.currentTimeMillis();
            System.out.println("\n\nNeural Network evaluation
time: "+(timefin-time));
            return c;
      }
}
```

Appendice B:

Pattern Recognition

Attraverso l'implementazione del seguente *script* si sono testate le prestazioni della tecnica di *pattern recognition*:

```
% Solve a Pattern Recognition Problem with a Neural Network
% Script generated by Marco Vecchi
% Created Mon Aug 06 14:05:57 CEST 2012
clc
clear
load glass_dataset;
inputs = glassInputs;
targets = glassTargets;

% Create a Pattern Recognition Network
hiddenLayerSize = 100;
net = patternnet(hiddenLayerSize);

% Choose Input and Output Pre/Post-Processing Functions
net.inputs{1}.processFcns =
{'removeconstantrows','mapminmax'};
net.outputs{2}.processFcns =
{'removeconstantrows','mapminmax'};

% Setup Division of Data for Training, Validation, Testing
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% For help on training function 'trainlm' type: help trainlm
net.trainFcn = 'trainlm';  % Levenberg-Marquardt

% Choose a Performance Function
net.performFcn = 'mse';  % Mean squared error
```

```
% Choose Plot Functions
net.plotFcns = {'plotperform','plottrainstate','ploterrhist',
'plotregression', 'plotfit'};

% Point Stop train
net.trainParam.min_grad=1e-4;
net.trainParam.goal=1e-4;
net.trainParam.max_fail=3;
net.trainParam.epochs=1e3;
net.trainParam.mu_max=1e3;
net.trainParam.mu=1;
net.trainParam.mu_dec=0.8;
net.trainParam.mu_inc=1.2;

% Train the Network
[net,tr] = train(net,inputs,targets);

% Test the Network
outputs = net(inputs);
errors = gsubtract(targets,outputs);
rmse=sqrt(mse(errors))
performance = perform(net,targets,outputs);

% Recalculate Training, Validation and Test Performance
trainTargets = targets .* tr.trainMask{1};
valTargets = targets  .* tr.valMask{1};
testTargets = targets  .* tr.testMask{1};
trainPerformance = sqrt(perform(net,trainTargets,outputs))
valPerformance = sqrt(perform(net,valTargets,outputs))
testPerformance = sqrt(perform(net,testTargets,outputs))

% Plots
figure, plotperform(tr)
figure, plottrainstate(tr)
figure, plotconfusion(targets,outputs)
```

Appendice C:

Spectrum in FDI

Script di Matlab

Il seguente *script* è stato implementato per testare e valutare la possibile applicazione dell'analisi spettrale nell'attività di FDI:

```
clc;
clear;
fs = 1024; % Sampling frequency
t = (0:fs-1)/fs; % One second worth of samples
freq = 1:length(t)/2 ;
P_noise = 0.2;
N = 500; % number of iteration

% Nominal behavior parameters
A1 = [1 0.6]; % Sinusoid amplitudes (row vector)
f1 = [60;200]; % Sinusoid frequencies (column vector)

% Fault behavior
A2 = [0.2];
f2 = [110];

% Treshold
k_max=7;
k_mean=3;

P_fault=zeros(1,length(P_noise)) + sum((A2.^2)/2);
SFNR=10*log10(P_fault./P_noise);

% Nominal behavior
xn = A1 * sin(2*pi*f1*t) + sqrt(P_noise) * randn(size(t));
Xn = fft(xn)/fs;
modX = abs(Xn);
X = 20*log10(abs(Xn(freq)));
```

```
for i=1:N
    % Measured nominal behavior
    yn = A1 * sin(2*pi*f1*t) + sqrt(P_noise) * randn(size(t));
    Yn = fft(yn)/fs;
    modY = abs(Yn);
    Y = 20 * log10(abs(Yn(freq)));

    % Measured fault behavior
    zn= A1 * sin(2*pi*f1*t) + sqrt(P_noise) * randn(size(t)) +
A2 * sin(2*pi*f2*t);
    Zn=fft(zn)/fs;
    modZ=abs(Zn);
    Z=20*log10(abs(Zn(freq)));

    % MSE Method
    square_error_ok = (modY-modX).^2;
    mse_ok(i) = mean(square_error_ok);
    square_error_fault = (modZ-modX).^2;
    mse_fault(i) = mean(square_error_fault);
    i=i+1;
end

% plot first (no fault) example
figure(1);subplot(2,1,1),plot(freq,X,'r-',freq,Y,'b--
');title('Spectrum (with NO fault)');
hleg = legend('Nominal','Real time');ylabel('PSD
[dB]');xlabel('Frequency [Hz]'); grid on;

subplot(2,1,2),plot(freq,square_error_ok(freq));title(strcat('
Residual (with NO fault) SFNR= ',int2str(SFNR(1))));
ylabel('Residual');xlabel('Frequency [Hz]');grid on;

% plot first faulty example
figure(2),subplot(2,1,1),plot(freq,X,'r-',freq,Z,'b--
');title('Spectrum (with fault)');
hleg = legend('Nominal','Real time');ylabel('PSD
[dB]');xlabel('Frequency [Hz]');grid on;

subplot(2,1,2),plot(freq,square_error_fault(freq));title('Resi
dual (with fault)');
ylabel('Residual');xlabel('Frequency
[Hz]');title(strcat('Residual (with fault)));grid on;
```

Codice Java

In codice Java© è stato realizzato l'algoritmo vero e proprio della valutazione spettrale:

```
public class Spectrum{
     private double[] PSD;

     public Spectrum(double[] x){
          this.PSD=toPSD(x);
     }

    // compute the FFT of x,assuming its length a power of 2
    private double[] toPSD(double[] x){
     int length=x.length;
     double alpha=1/length;
     Complex[] s=new Complex[x.length];
          for(int k=0;k<x.length;k++){
               s[k]=new Complex(x[k]);
          }
          Complex[] X=fft(s);
          double[] PSD = new double[x.length/2];
          double[] PSD_dB=new double[x.length/2];
          for(int k=0;k<x.length/2;k++){
               X[k]=X[k].times(alpha);
               PSD[k]=X[k].abs();
               PSD_dB[k]=20*Math.log10(PSD[k]);
          }
          return PSD;
    }

    public double[] evaluate(double[] signal){
     long time = java.lang.System.currentTimeMillis();
     double[] s=toPSD(signal);
     if (s.length != this.PSD.length) { throw new
RuntimeException("Dimensions don't agree"); }
     double[] residual=new double[s.length];
     for (int i=0;i<s.length;i++){
          residual[i]=Math.pow(s[i]-this.PSD[i],3);
     }
     long timefin = java.lang.System.currentTimeMillis();
```

```
            System.out.println("Spectrum evaluation time:
"+(timefin-time));
            return residual;
    }

     //radix 2 Cooley-Tukey FFT
     public static Complex[] fft(Complex[] x) {
        int N = x.length;

        // base case
        if (N == 1) return new Complex[] { x[0] };

        if (N % 2 != 0) { throw new RuntimeException("N is not
a power of 2"); }

        // even terms
        Complex[] even = new Complex[N/2];
        for (int k = 0; k < N/2; k++) {
            even[k] = x[2*k];
        }
        Complex[] q = fft(even);

        // odd terms
        Complex[] odd  = even;  // reuse the array
        for (int k = 0; k < N/2; k++) {
            odd[k] = x[2*k + 1];
        }
        Complex[] r = fft(odd);

        // combine
        Complex[] y = new Complex[N];
        for (int k = 0; k < N/2; k++) {
            double kth = -2 * k * Math.PI / N;
            Complex wk = new Complex(Math.cos(kth),
Math.sin(kth));
            y[k]       = q[k].plus(wk.times(r[k]));
            y[k + N/2] = q[k].minus(wk.times(r[k]));
        }
        return y;
    }
```

```
    // compute the inverse FFT of x[], assuming its length is
a power of 2
    public static Complex[] ifft(Complex[] x) {
        int N = x.length;
        Complex[] y = new Complex[N];
        // take conjugate
        for (int i = 0; i < N; i++) {
            y[i] = x[i].conjugate();
        }

        // compute forward FFT
        y = fft(y);

        // take conjugate again
        for (int i = 0; i < N; i++) {
            y[i] = y[i].conjugate();
        }

        // divide by N
        for (int i = 0; i < N; i++) {
            y[i] = y[i].times(1.0 / N);
        }
        return y;
    }
    // compute the circular convolution of x and y
    public static Complex[] cconvolve(Complex[] x, Complex[]
y) {
        // should probably pad x and y with 0s so that they
have same length
        // and are powers of 2
        if (x.length != y.length) { throw new
RuntimeException("Dimensions don't agree"); }

        int N = x.length;

        // compute FFT of each sequence
        Complex[] a = fft(x);
        Complex[] b = fft(y);

        // point-wise multiply
        Complex[] c = new Complex[N];
        for (int i = 0; i < N; i++) {
            c[i] = a[i].times(b[i]);
        }
```

```
        // compute inverse FFT
        return ifft(c);
    }

    // compute the linear convolution of x and y
    public static Complex[] convolve(Complex[] x, Complex[] y)
{
        Complex ZERO = new Complex(0, 0);

        Complex[] a = new Complex[2*x.length];
        for (int i = 0; i < x.length; i++) a[i]=x[i];
        for (int i = x.length; i < 2*x.length; i++) a[i]=ZERO;
        Complex[] b = new Complex[2*y.length];
        for (int i = 0; i < y.length; i++) b[i] = y[i];
        for (int i = y.length; i < 2*y.length; i++) b[i]=ZERO;
        return cconvolve(a, b);
    }
}
```

Appendice D:

Database Import

Per importare dal database i dati raccolti e quindi poterli graficare è stato implementato il seguente *script*:

```
clc
clear
logintimeout(5);
name='tmc_datalogging';
username='tmc_datalogger';
password='tmc_datalogger';
javaaddpath
'C:\Users\Marco\workspace\OptimaDataLogger\lib\mysql-
connector-java-5.1.22-bin.jar';
conn = database(name, username, password,
'com.mysql.jdbc.Driver',
'jdbc:mysql://192.168.1.110:3306/tmc_datalogging');
current = fetch(conn, 'select value_long,timestamp_millisec
from dataset where id_tag
=''PLC_Id23.Buffering.MC08_Lift_Current''');
refvelocity = fetch(conn, 'select value_long from dataset
where id_tag =''PLC_Id23.Buffering.MC08_Lift_RefVelocity''');
velocity = fetch(conn, 'select value_long from dataset where
id_tag =''PLC_Id23.Buffering.MC08_Lift_Velocity''');
dim=size(velocity);
Lift_Current=zeros(1,dim(1));
Lift_RefVelocity=zeros(1,dim(1));
Lift_Velocity=zeros(1,dim(1)); timestamp=zeros(1,dim(1));
for i=1:dim(1)
    Lift_Current(i)=current{i,1};
    Lift_RefVelocity(i)=refvelocity{i,1};
    Lift_Velocity(i)=velocity{i,1};
    timestamp(i)=current{i,2};
end
figure,plot(1:dim(1),Lift_Current,'b',1:dim(1),
Lift_RefVelocity,'r',1:dim(1),Lift_Velocity,'k');
hleg = legend('Velocity','RefVelocity','Velocity');grid on;
```

Bibliografia

[1] Ron J. Patton, Paul M. Frank and Robert N. Clark. *Issues of Fault Diagnosis for Dynamic Systems.* s.l. : Springer, 2000.

[2] Nikoukhah, Stephen L. Campbell and Ramine. *Auxiliary Signal Design for Failure Detection.* s.l. : Princeton University, 2004.

[3] Carlson, Carl S. *Annual RELIABILITY and MAINTAINABILITY Symposium*. 2012.

[4] Nigel Palastanga, Derek Field, Roger Soames. Sistema nervoso. *Anatomia del movimento umano. Struttura e funzione.* s.l. : Elsevier, 2007.

[5] Rojas, Raùl. The Biological Paradigm. *Neural Networks, A Systematic Introduction.* Berlin : Springer-Verlag, 1996.

[6] Riedmiller, Martin. *Machine Learning: Multi Layer Perceptrons.* University Freiburg : s.n.

[7] P. Subbaraj, B. Kannapiran. *Artificial Neural Network Approach for Fault Detection in Pneumatic Valve in Cooler Water Spray System.* International Journal of Computer Applications, November 2010, Vol. 9.

[8] prof. Davide Maltoni http://bias.csr.unibo.it/maltoni/v&r/DispensePDF/4_V&R_RetiNeurali.pdf. [Online]

[9] Brugnano, prof. L. Metodi Numerici per l'Ottimizzazione. [Online] http://www.mascanc.net/~max/uni/mno/relazione/relazione.pdf.

[10] R. Zayani, R. Bouallegue, D. Roviras. *Levenberg-Marquardt Learning Neural Network For Adaptive Predistortion For Time-Varying HPA With Memory In OFDM Systems.* Losanna, Switzerland : 16th European Signal Processing Conference, 2008.

[11] Rojas, Raùl. The Backpropagation Algorithm. *Neural Networks, A Systematic Introduction.* Berlin : Springer-Verlag, 1996.

[12] Wasilewska, Anita. *Neural Networks.* University of New York.

[13] R. Fernandes, D. Silva, L. de Oliveira, A. Neto. *Faults Detection and Isolation Based on Neural Networks Applied to a Levels Control System.* Orlando, Florida, USA : Proceedings of International Joint Conference on Neural Networks, 2007.

[14] H. Kim, S. Fok, K. Fregene, D. Lee,T. Oh and D. Wang. *Neural Network-Based System Identification and Controller Synthesis for an Industrial Sewing Machine.* International Journal of Control, Automation, and Systems, 2004, Vol. 2.

[15] Mark Hudson Beale, Martin T. Hagan,Howard B. Demuth. *Neural Network Toolbox™.* 2012.

[16] Frank, H. Schneider and P. *Observed-Based Supervision and Fault Detection in Robots Using Nonlinear and Fuzzy Logic Residual Evaluation.* IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, 1996, Vol. 4.

[17] Li-Ping He, Hong-Zhong Huang, Ming J. Zuo. *Fault Tree Analysis Based on Fuzzy Logic.*

[18] MathWorks. *Fuzzy Logic Toolbox™ User's Guide.* 2012.

[19] Chiani, Marco. *Notes on Detection and Estimation.* 2003.

[20] Indian Institute of Technology Delhi. Fault Identification and Monitoring.

[21] Khalaf Salloum Gaeid, Hew Wooi Ping, Mustafa Khalid, Atheer Lauy Salih. *Fault Diagnosis of Induction Motor Using MCSA and FFT.* Kuala Lumpur, Malaysia : 2011.

[22] Giovanni Betta, Massimo D'Apuzzo, Consolatina Liguori, and Antonio Pietrosanto. *An Intelligent FFT-Analyzer.* 1998.

[23] Groover. *Automation, Production Systems, and Computer Integrated Manufacturing.* Prentice Hal, 2007.

[24] Rossi, Carlo. *Reti di comunicazione per sistemi di controllo.*

[25] Canini Gabriele, Fantuzzi Cesare. *Controllo del moto per macchine automatiche.* Pitagora, 2003.

[26] *PLC standard programming languages: IEC 1131-3.* Rockwell Software Inc. USA: 1996.

[27] prof. Carlo Rossi, *Azionamenti elettrici, catterisitche dei motori.*

[28] Enea. *Applicazione di sistemi di accumulo elettrici in ambito industriale.* 2009.

[29] Lutz, Peter. Introduction to SERCOS interface. *http://www.sercos.com/literature/pdf/workshop4_MCS04-SCS-plu1.pdf.* [Online] 2000.

[30] Li Zheng, Hiroyuki Nakagawa, *OPC (OLE for Process Control) Specification and its Developments.*. Osaka (Japan) , 2002.

[31] ABB. OPC Server. *http://www05.abb.com/global/scot/scot209.nsf/veritydisplay/8589f307e0be08b2c125723300392640/$file/2cdc125023m0201.pdf.* [Online]

[32] Schneider electric. OPC Server 3 - Installation and Usage. *http://www2.schneider-electric.com/resources/sites/SCHNEIDER_ELECTRIC/content/live/FAQS/136000/FA136910/en_US/SoMachine%20and%20OPC%20server.pdf.* [Online]